НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено

Завідувач кафедри

_____ О.В. Коваль
(підпис) (ініціали, прізвище)

“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 “ Інженерія програмного забезпечення “

на тему: Кластеризація даних, що збираються з відібраних джерел науково-технічної інформації

Виконав (-ла): студент (-ка) 4 курсу, групи ТІ-61

Крамар Іван Ігорович _____ _____
(прізвище, ім’я, по батькові) (підпис)

Керівник зав. кафедри, доцент, канд.техн.наук, Коваль О.В. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент ст.наук.співробітник інституту проблем реєстрації інформації НАН України Сенченко В.Р._____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший , бакалаврський

Напрям підготовки 121 “ Інженерія програмного забезпечення “

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту

Крамару Івану Ігоровочу
(прізвище, ім’я, по батькові)

1. Тема роботи Кластеризація даних, що збираються з відібраних джерел науково-технічної інформації

керівник роботи Коваль Олександр Васильович, канд.техн.наук, доцент
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 201__р. № __

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи: мова Python на платформі macOS

4.Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити):
1) Здійснити постановку задачі кластеризації науково-технічних даних. 2) Розглянути методи, етапи та алгоритми інтелектуального аналізу текстової інформації. 3) Розглянути алгоритми кластерного аналізу та провести їх порівняння. 4) Спроектувати систему кластерного аналізу науково-технічних даних. 5) Реалізувати систему кластерного аналізу науково-технічних даних.

5. Перелік ілюстративного матеріалу

_Актуальність; Постановка задачі; Схема кластерного аналізу; Вибір алгоритму кластерізації; K-mean кластеризація; Вибір мови програмування; Розглянемо на прикладі; Візуалізація результату; Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання "_11_" _жовтня__2019_р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Вивчення та аналіз задачі	12.03.2020 – 04.06.2020	
2	Розробка архітектури та загальної структури системи	12.03.2020 – 04.06.2020	
3.	Розробка структур окремих підсистем	12.03.2020 – 04.06.2020	
4.	Програмна реалізація системи	12.03.2020 – 04.06.2020	
5.	Оформлення пояснювальної записки	12.03.2020 – 04.06.2020	
6.	Захист програмного продукту	11.05.2020 – 15.05.2020	
7.	Передзахист	09.06.2020 – 12.06.2020	
8.	Захист	15.06.2020 – 19.06.2020	

Студент _____ _Крамар І.І._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _Коваль О.В._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Обсяг дипломної роботи складає 94 сторінок, 5 розділів, 39 рисунків, 11 таблиць, 22 джерела в переліку посилань.

Метою роботи є застосування кластеризації науково-технічних даних не тільки для наглядного представлення об’єктів, але і для розпізнавання нових. Метою кластеризації документів є автоматичне виявлення груп семантично схожих документів серед заданої фіксованої множини. Групи формуються тільки на основі попарної схожості описів документів, і ніякі характеристики цих груп не задаються заздалегідь.

Для видалення неінформативних слів розглянуто методи: видалення стоп-слів, стеммінг, N-діаграми, приведення регістра.

Для виділення ключових слів та класифікації результатів використано наступні методи: словниковий, статистичний та побудований на основі Y-інтерпретації закону Бредфорда, TF-IDF міра, F-міра та метод лакричних шаблонів.

Для реалізації системи кластерного аналізу науково-технічних даних обрано високорівневу мову програмування Python, реалізація інтерпретатора 2.7. Даний програмний код читається легше, його багаторазове використання і обслуговування виконується набагато простіше, ніж використання програмного коду на інших мовах.

Ключові слова: кластеризація, видалення стоп-слів, стеммінг, N-грами, К-середніх, ієрархічна класифікація, алгоритм DBSCAN.

ABSTRACT

The volume of the thesis is 94 pages, 5 sections, 39 forms, 11 tables, 22 points in the link list.

The aim of the work is to use the clustering of scientific and technical data not only for the visual representation of objects, but also for the recognition of new ones. The purpose of document clustering is to automatically detect groups of semantically similar documents among a given fixed set. Groups are formed only on the basis of pairwise similarity of document descriptions, and no characteristics of these groups are set in advance.

Methods for deleting uninformative words are considered: deletion of stop words, stemming, N-diagrams, case reduction.

The following methods were used to highlight keywords and classify the results: dictionary, statistical and based on the Y-interpretation of Bradford's law, TF-IDF measure, F-measure and the method of licorice patterns.

Python programming language was chosen to implement the system of cluster analysis of scientific and technical data, a high-level, the implementation of the interpreter 2.7. This program code is easier to read, its reuse and maintenance is much easier than using program code in other languages.

Keywords: clustering, stop word deletion, stemming, N-grams, K-means, hierarchical classification, DBSCAN algorithm.

ЗМІСТ

ВСТУП

На даний час глобальний процес автоматизації зачепив практично всі сфери людської діяльності, включаючи наукову галузь. Сучасні інформаційні технології дозволяють накопичувати досить велику кількість інформації, що є характерним, зокрема і для наукових та навчальних закладів. Науково-технічні дані накопичуються роками та досить часто є різнорідними (невпорядкованими). Варто зазначити, що особливістю таких даних є багато розмірність, що ускладнює процес аналізу даних. Також дані можуть бути представлені не тільки в числовому вигляді, але і у категорійному вигляді. На сьогоднішній день досить відомою технологією аналізу даних Datamining.

Datamining – це процес виявлення у необроблених даних раніше невідомих нетривіальних, практично корисних і доступних інтерпретацій знань, необхідних для прийняття рішень у різних сферах діяльності. Datamining вирішує ряд задач, які пов’язані з інтелектуальним аналізом даних. Однією із задач, які вирішує Datamining є кластеризація – процес, що дозволяє групувати деякий набір об’єктів за схожим набором параметрів[1]. Іншими словами, з’являється можливість розподілити накопичені дані відповідних закладів по певних групах, за деякими схожими критеріями.

На разі існує велика кількість алгоритмів кластеризації, які відрізняються, щонайменше, набором вхідних параметрів і в залежності від обраних параметрів результати кластеризації можуть відрізнятись. Виникає складність в оцінці результату кластеризації. Один із способів оцінити результат кластеризації –розташувати об’єкти та центри кластерів на координатній площині, але тут виникає інша проблема – у випадку, коли дані багато розмірні відпадає можливість розташування їх на координатній площині. В такому випадку необхідна деяка методика оцінки результатів кластеризації.

1 ОГЛЯД ПРОБЛЕМИ АНАЛІЗУ НАУКОВО-ТЕХНІЧНИХ ДАНИХ

Сучасний рівень розвитку інформаційних технологій дозволяє різним установам накопичувати та зберігати дані у базах даних, де наукові та навчальні заклади не є виключенням. З часом кількість накопичених даних зростає та досягає великих розмірів. Через достатньо велику кількість інформації – співробітник може ознайомитись лише з деякою її частиною, тому є необхідним знайти спосіб опрацювання інформації значно ефективнішим шляхом. Також варто зазначити, що особливістю науково-технічних даних є представлення їх не завжди в числовому вигляді, що ускладнює процес обробки даних машиною. На даний час для отримання корисної інформації з великого набору даних використовують технологію – Datamining.

Datamining – це технологія, яка вивчає процес отримання нових, реальних та раніше невідомих знань в базах даних. Однією із задач технології Datamining є процес кластеризації. Процес кластеризації дозволяє розглядати великі набори даних та різко скорочувати їх, робити їх компактними та наглядними. Задача кластеризації полягає в розподіленні деякої множини об’єктів на групи схожих об’єктів, що називаються кластерами. Результатом процесу кластеризації являється набір кластерів, які містять в собі схожі елементи вхідної множини елементів[1].

На разі існує велика кількість алгоритмів кластеризації, які відрізняються набором вхідних параметрів і в залежності від обраних параметрів результати кластеризації можуть відрізнятись. В контексті задачі кластеризації науково-технічних даних виникає проблема, яка полягає у виборі алгоритму кластеризації: алгоритм кластеризації необхідно вибрати таким чином, щоб розбиття на кластери було якомога правильним.

Однією із найпоширеніших проблем алгоритмів кластеризації є те, що більшість алгоритмів кластеризації очікують вхідним параметром число

кластерів і зазвичай кількість кластерів заздалегідь невідома[2]. І тому потрібно використовувати деякі емпіричні правила, для вибору оптимального числа кластерів. Що ж стосується задачі кластеризації науково-технічних даних, то тут у більшості випадків кількість кластерів заздалегідь відома, оскільки науковий співробітник точно знає як класифікується та чи інша інформація.

Задача кластеризації набуває значно вагомішого характеру, якщо застосувати кластери не тільки для наглядного представлення об'єктів, але і для розпізнавання нових. Кожен новий об'єкт відноситься до того кластеру, приєднання до якого найкращим чином буде задовольняти критерій якості кластеризації. Тому, можемо припустити, що поведінка об'єкту може бути схожою із поведінкою інших об'єктів, які знаходяться в кластері.

1.1 Постановка задачі кластеризації науково-технічних даних

Нехай відома множина об'єктів НТІ I, кожен із яких представлений деяким набором атрибутів. Потрібно побудувати множину кластерів C, та відображення F множини I на множину C, тобто $F{:}I{\rightarrow}C$. При цьому об'єкти, що належать одному кластеру – повинні бути схожі між собою, а ті що належать до різних кластерів – повинні істотно відрізнятись. Відображення F, задає модель даних, яка є рішенням[2]. Якість рішення задачі визначається кількістю вірно кластеризованих об'єктів. Визначимо множину наступним чином

$$I=\{i_1,i_2,...i_n\} \tag{1.1}$$

де i_n–об'єкт, що досліджується.

В свою чергу кожен із досліджуваних об'єктів характеризується набором параметрів:

$$i_n=\{x_1,x_2,...x_m\}. \tag{1.2}$$

Кожна змінна в свою чергу x_m–може приймати значення з деякої множини:

$$x_m=\{v^1{}_m,v^2{}_m,...\}. \quad (1.3)$$

Задача кластеризації полягає в побудові множини кластерів:

$$C=\{c_1,c_2,...,c_g\}, \quad (1.4)$$

c_g–кластер, який містить схожі об'єкти із множини

$$I:c_g=\{i_n,i_p|\ i_n\epsilon I,\ i_p\epsilon I\ \mathrm{I}d(i_n,i_p)<\sigma\}, \quad (1.5)$$

де σ-величина, що визначає міру близькості для включення об'єктів в один кластер;

$d(i_n,i_p)$–міра близькості між об'єктами, яка називається відстанню.

Невід'ємне значення $d(i_n,i_p)$ – називається відстанню між елементами i_n та i_p якщо виконуються наступні умови:

$-d(i_n,i_p)\geq 0$, для всіх i_n та i_p, (1.6)

$-d(i_n,i_p)=0$,тоді і тільки тоді, коли $in=ip$, (1.7)

$-d(i_n,i_p)=d(i_p,i_n)$, (1.8)

$-d(i_n,i_p)\leq d(i_p,i_r)+d(i_r,i_p)$. (1.9)

Якщо відстань $d(i_n,i_p)$ менше деякого значення σ, то можна сказати, що елементи розташовані близько один до одного та знаходяться в одному кластері. В іншому випадку елементи знаходяться в різних кластерах. Рішенням задачі є розбиття елементів на кластери, яке повинно задовольняти деякому критерію. Цей критерій може являти собою деякий функціонал, який називають цільовою функцією.

ВИСНОВКИ ДО РОЗДІЛУ 1

1. Здійснено огляд проблеми аналізу науково-технічних даних. Для вирішення даної проблеми обрано технологію DataMining, а саме, одна із складової даної технології – кластеризація даних.

2. Виявлено основні проблеми кластеризації науково-технічних даних: багаторозмірність даних, можливе категоріальне представлення даних, вибір методу кластеризації та оцінка результату.

3. Здійснено постановку задачі кластеризації науково-технічних даних.

2 ОГЛЯД ІСНУЮЧИХ ТЕХНОЛОГІЙ ТА РІШЕНЬ

2.1 Основні задачі

Спостерігається так званий «інформаційний вибух». Тобто тяжко знайти джерела інформації, ця проблема перетворилася на проблему вибору або фільтрації потрібної інформації серед інформаційного шуму.

З’явилася проблема створення систем з пошуку інформації та структуруванні цієї інформації в текстах. Для вирішення виник напрямок інтелектуального аналізу текстів - Text Mining. Його відносять до розділу Data Mining.

Виявлення інформації в тексті - це нетривіальний процес виявлення нових, потенційно корисних шаблонів в неструктурованих текстах.

Як ми бачимо, з визначення Data Mining, відрізняється тільки новими словами "неструктуровані тексти". Під такою інформацією розуміється набір документів, що представляють собою логічно з'єднаний текст без будь-яких обмежень на його структуру. Як приклад таких документів є: Web-сторінки, нормативні документи, електронна пошта і т. п. У загальному випадку такі документи бувають складними і великими та можуть включати не тільки текст, а й графічну інформацію. Документи, що використовують мову XML (*eXtensible MarkupLanguage*), мову узагальненої розмітки SGML (*Standard Generalised Markup Language*) та інші подібні тексти, прийнято називати напівструктурованими документами. Вони теж можуть бути оброблені методами Text Mining.

На даний плин часу в літературі описано багато прикладних задач, що вирішуюся шляхом застосування методів аналізу текстових документів. Це і класичні задачі Data Mining: кластеризація, класифікація, і характерні тільки для текстових документів завдання: автоматичне анотування, витяг ключових понять та ін.

Класифікація (*classification*) – це стандартна задача Data Mining. Її метою є визначення для документа декількох заздалегідь заданих категорій, до яких цей документ має відношення. Особливістю задачі класифікації є припущення, що певна кількість документів, що класифікуються, не містить непотрібних тем, тобто кожен з документів відповідає будь-якій заданій категорії. Окремим випадком класифікації є задача визначення тематики документа.

Метою **кластеризації** (*clustering*) документів є автоматичне виявлення груп схожих документів серед фіксованої множини яку ми задали. Групи створюються тільки на основі попарної схожості опису документу.

Автоматичне анотування (*summarization*) дозволяє зменшити текст не змінюючи його зміст. Вирішення цієї роботи зазвичай регулюється користувачем за допомогою визначення кількості речень. Результат включає в себе лише потрібні нам речення в тексті.

Метою **виділення ключових понять** (*feature extraction*) є знаходження фактів і відношень в тексті. Майже у всіх випадках такими поняттями є іменники і загальні назви: назви організацій, імена, прізвища людей та ін.

Навігація по тексту (*text-base navigation*) дозволяє користувачам переходити по документам відповідно значущих термінів і тем. Це виконується за допомогою ідентифікації основних понять і відношень між ними.

Виявлення (*detection*) дозволяє виявити в потоці даних нову цікаву інформацію, таку як конструкції, асоціації, моделі, зміни, аномалії.

Системи автоматичних відповідей (*automatical answers systems*) дає відповіді на питання деяких користувачів, що задаються на природній мові.

За базу інформації в цих системах передбачається використовувати ресурси Internet, оброблені новітніми засобами аналізу текстів.

2.2 Вимоги до системи

1. **Точність** (*accuracy*) - алгоритми обробки НЕ гарантують отримання тільки коректних результатів, тому дизайн системи повинен враховувати це і дати можливість для підвищення точності, наприклад, переходу до використання іншого алгоритму;

2. **Продуктивність** (*productivity*) – сили які були витрачені на створення ПМ-додатків часто вищі, аніж для більшості областей розробки ПЗ у зв'язку з браком знань про існуючі ресурси;

3. **Гнучкість** (*flexibility*) - як все програмне забезпечення ПМ-системи мають бути гнучкими, вони повинні розуміти різні формати даних та мати можливість використовуватись для різних завдань;

4. **Стійкість** (*robustness*) - система повинна бути працездатна в різних ситуаціях, містити обробку різних помилок алгоритмів і ситуацій;

5. **Масштабованість** (*scalability*) - для варіантів обробки великих обсягів даних система повинна запросто збільшувати продуктивність наприклад, -використання розподіленої схеми;

6. **Мультимодальність** (*multimodality*) - різні лінгвістичні компоненти використовують при аналізі деякі свої інформаційні аспекти, тому система повинна підтримувати доступ до них;

7. **Розрідженість даних** (*data sparseness*) - багато алгоритмів при роботі з ПМ-текстами просять наявність певних даних для навчання, їх побудова може являти собою проблему;

2.3 Загальні етапи та підходи

Процес аналізу текстових документів можна представити як послідовність кроків (рис. 2.1).

1. *Пошук інформації*. На першому кроці нам потрібно знайти, які документи повинні бути проаналізовані і забезпечити доступ до них. Зазвичай, користувачі можуть вибрати документи, що проаналізуються самі - тобто вручну, але якщо кількість документів буде велика то необхідно використовувати варіант автоматизованого збору за певними критеріями.

2. *Попередня обробка документів*. На цьому кроці виконуються необхідні перетворення з документами для того щоб показати їх у вигляді, з яким можуть працювати методи Text Mining. Потрібні ці перетворення для видалення зайвих слів. Більш детально ми про них поговоримо далі.

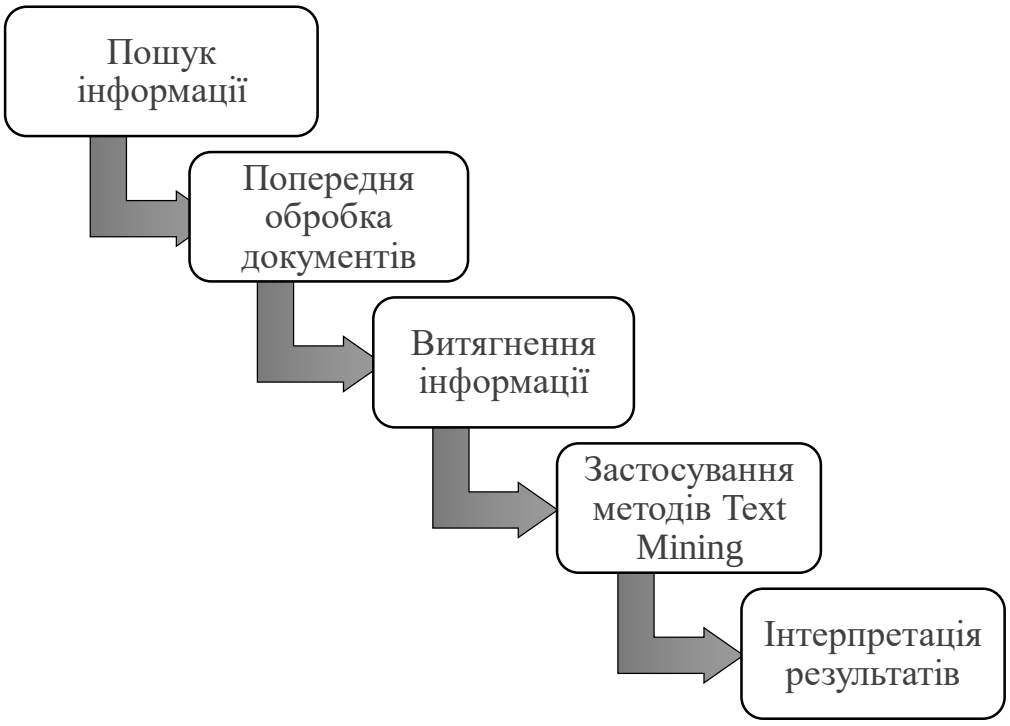


Рисунок 2.1 – Етапи Text Mining

3. *Витягнення інформації*. Витяг інформації з наших документів означає відображення в них ключових понять, з якими далі будемо виконувати аналіз. Цей етап є дуже важливим і буде описаний далі.

4. *Застосування методів* Text Mining. На цьому етапі витягуються шаблони і відношення, які є в текстах. Це основний крок у процесі аналізу текстів, і практичні задачі, що вирішуються, описуються далі.

5. *Інтерпретація результатів*. Останній крок у процесі отримання інформації передбачає інтерпретацію отриманих результатів. Зазвичай, інтерпретація полягає в їх візуалізації в графічному виді.

2.4 Попередня обробка

Однією з найбільших проблем аналізу текстів є велика кількість непотрібних слів, а також різні форми слів. Якщо аналізувати всі такі слова, то час затрачений на пошук нової інформації буде дуже великий. Не всі слова в тексті несуть корисну інформацію. Крім того різні слова (синоніми і т. п.) означають однакові поняття. Тому, видалення непотрібних слів, а також приведення близьких за змістом до єдиної форми сильно зменшують час аналізу текстів.

Зазвичай використовують такі методи видалення непотрібних слів і підвищення строгості текстів:

- ✓ **видалення стоп-слів**. Стоп-словами називаються слова, які є не основними і несуть малу кількість інформації про документ. Як правило складаються списки таких слів, і в процесі обробки вони видаляються з тексту. Прикладом таких слів є слова і артиклі, типу: "так як", "Крім того" і т. п.;
- ✓ **стеммінг** - морфологічний пошук. Полягає в зміні кожного слова до його нормальної формі. Нормальна форма прибирає схиляння, множинну форму і т. п. Наприклад, слова "розміщення" і "розміщений" мають бути перетворені до нормальної форми "розмістити". Алгоритми розбору враховують особливості і через це є мовно-залежними алгоритмами;

2.5 Зниження розмірності

Описані нижче алгоритми використовуються для зниження розмірності, що дозволяє ефективно візуалізувати багатовимірні компоненти. З багатовимірної змінної ми намагаємося отримати нову змінну, яка буде лежати в дво- або тривимірному просторі і зберігати закономірності вихідної змінної.

2.5.1 Алгоритм t-SNE

Назва алгоритму є скорочення з повного варіанту t-distributed stochastic neighbor embedding. На українську мову можна його спробувати перевести як «t-розподілене впровадження сусідів». Даний метод відноситься до методів множинного навчання ознак.

2.5.1.1 Математичний опис алгоритму

Будемо будувати бієктивне відображення, тобто кожна точка буде представляти один екземпляр (рядок) вихідного набору текстів. Щоб візуалізація могла відображати поділ на класи, ми хочемо, щоб точки, що належать одному класу, розташовувалися поруч, що і покаже формула (2.1):

$$p_{j|i} = \frac{\exp\left(\frac{-\left|x_i - x_j\right|^2}{2\sigma_i^2}\right)}{\sum_{k \neq j}^{n} exp\left(\frac{-\left|x_i - x_j\right|^2}{2\sigma_i^2}\right)} \qquad (2.1)$$

Формула (2.1), що основана на гаусовському розподілі навколо точки σ_i із заданою дисперсією визначає умовне схожість двох точок. Дисперсії обчислюються таким чином, щоб точки, розташовані в областях з малою щільністю, мали велику дисперсію, ніж точки, розташовані в областях з

великою щільністю. Для цього використовується оцінка перплексії (2.2). Зазвичай ця оцінка використовується для порівняння імовірнісних моделей, при цьому низьке значення перплексії означає, що розподіл ймовірностей (закон, що описує область значень випадкової величини і ймовірності їх результату) добре працюють на етапі передбачення. У нашому випадку можна її інтерпретувати як згладжену оцінку ефективної кількості «сусідів» для конкретної точки.

$$P_{erp}(P_i) = 2^{H(P_i)} \tag{2.2}$$

Для цього обчислюється ентропія Шеннона в бітах (2.3):

$$P_i = -\sum_j p_{j|i} \log_2 p_{j|i} \tag{2.3}$$

Подібність двох точок буде визначатися через формулу (2.4) умовної подібності двох точок:

$$p_{j|i} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{2.4}$$

На основі цієї формули обчислюється матриця подібності для вихідних даних. Дана матриця є постійною.

На відміну від матриці подібності вихідних даних, матриця подібності для відображення залежить від точок відображення. Близькість цих двох матриць буде нам доводити, що схожі вихідні точки відображаються в також схожі точки.

При визначенні матриці подібності для точок відображення (2.5) замість гаусовського розподілу використовується розподіл Стьюдента з одним ступенем свободи або розподіл Коші. Причина даної заміни полягає в наступному: відстань між парою точок в просторі відображення, які відповідають парі середньовіддаленних точок у вихідному просторі, має бути набагато більше, ніж відстань, яку можна отримати за допомогою гаусовського розподілу. А метод намагається відтворити однакові відстані в обох просторах, і виникає проблема «скупченості». Завдяки важким «хвостам» розподіленні Стьюдента дисбаланс у розподіленні відстаней для сусідів точок скомпенсований (2.5).

$$q_{ij} = \sum_{k \neq i}^{f(|x_i - x_j|)} f\left(|x_i - x_j|\right), \text{де } f(y) = \frac{1}{1+y^2} \quad (2.5)$$

Алгоритм зводить до мінімуму відстань між матрицю подібності, мінімізуючи відстань між p_{ij} та q_{ij} (2.6):

$$\mathrm{KL(P||Q)} = \sum_{i,j} log_{q_{ij}}^{p_{ij}} \quad (2.6)$$

Для мінімізації цієї дистанції використовується градієнтний спуск. Градієнт можна зрозуміти як суму всіх сил, які прикладені до точки відображення і. У формулі (2.7) u_{ij} позначає один вектор, який йде від yj до yi

$$\frac{\partial \mathrm{KL(P||Q)}}{\partial y_i} = 4 \sum_i \left(p_{ij} - q_{ij}\right) g\left(|x_i - x_j|\right) u_{ij}, \text{ де } g(y) = \frac{1}{1+y^2} \quad (2.7)$$

2.5.1.2 Фізична аналогія

Цей алгоритм може бути інтерпретований з фізичної точки зору наступним чином. Ми вважаємо, що всі точки отриманого відображення з'єднані пружинами з різною жорсткістю.

Жорсткість пружин залежить від величини $p_{ij} - q_{ij}$, яка позначає різницю між подібністю пари точок вихідних даних і подібністю пари точок відображення.

Градієнт — це результуюча сила, яка діє на точці в просторі відображення. Ми відпускаємо систему, і вона зміниться, поки стан рівноваги не буде досягнуто. Відображення на цьому етапі буде те, що ми прагнемо.

2.6 Методи видалення стоп слів

У цій дипломній роботі проаналізовано 3 методи видалення стоп-слів: статистичний на основі поєднання слів із текстів, зі словником, та по основі Y-інтерпретації закону Бредфорда.

2.6.1 Словниковий

Цей метод являється у використанні готового списку стоп-слів. Перелік стоп-слів складається вручну - це є першим мінусом цього методу, адже це дуже довго. Іншим мінусом можна виділити малу універсальність словника, з чого слідує не повне вилучення стоп-слів, тому що цей список є досить великим і для багатьох варіантів текстів.

Перевагою цього методу є точність, тобто при використанні він не видалить слова, які мають смислове навантаження в тексті.

2.6.2 Статистичний на основі об’єднання

Метод являється в аналізі дуже об’ємної кількості текстів з предметної області, та вибірка з текстів списку найбільш повторюваних слів. Плюсом такого методу полягає в автоматизації і те що словник стоп-слів потрібно зробити лише один раз і далі не треба створювати його для всіх текстів. Але нажаль не враховуються особливості тексту. Може виникнути ситуація, при якій будуть вилучені слова, котрі згідно зі статистикою з’явилися в словнику стоп-слів.

2.6.3 За Y-інтерпретацією закону Бредфорда

Закон Бредфорда - емпіричне правило розподілу публікацій з виданням, згідно з якою в списку наукових журналів, розташованих в порядку убування

числа статей по заданому питанню, можна виділити три зони, що містять рівне число статей по заданому питанню.

Ці три зони відрізняються кількістю складових їх журналів:

- в першу зону (зону ядра) входять профільні журнали, безпосередньо присвячені заданому питанню;
- в другу зону входять журнали, частково присвячені заданому питанню;
- в останню третю зону входять журнали, тематика яких далека від заданого питання.

Згідно із законом Бредфорда для кожної тематичної області існує коефіцієнт кратного збільшення кількості журналів у кожної наступної зони.

Математичне формула закону виглядає так:

$$S_1 + qS_1 + q_2S_1 = S \qquad (2.13)$$

де: S_1 – перша зона (зона «ядра»);

qS_1 – друга зона

q_2S_1 – третя зона

S – загальна кількість публікацій

Y-інтерпретація закону Бредфорда - інтерпретація, запропонована В.А. Яцко, що дозволяє обчислювати порогові рівні при виділенні підмножин з безлічі на основі методики зонального аналізу.

Якщо розглядати аналіз тексту то суть полягає в поділі на 3 зони:

1. Зона J0 - зона службових слів або стоп-слова.
2. Зона J1 - слова, що представляють головний зміст тексту.
3. Зона J2 - слова, що рідко зустрічаються в тексті.

2.7 Ключові слова

Ключові слова - це лексичні групи, що пояснюють зміст документа. Автоматичне виділення ключових слів є обов’язковим етапом зчитування тексту в таких додатках, як системи автоматичного анотування, пошуку,

реферування і т. д. Але, незважаючи на велику кількість досліджень, автоматичне завантаження ключових слів являє собою досить суттєву проблему, яка до цього часу не вирішена. При всіх методиках алгоритм виділення ключових слів не унікальний та складається з етапів: 1) створення «кандидатів» в ключові слова та 2) фільтрації цієї множини для отримання списку ключових слів.

Основними виділяють 3 групи методів виділення ключових слів: статистичні, лексичні та гібридні.

2.7.1 Статистичні методи виділення ключових слів

Статистичні методи основуються на численних даних про періодичність зустрічання слова в тексті.

Перевагами статистичних методів є універсальність відсутність необхідності в процедурах побудови лінгвістичних баз знань, їх дуже просто реалізувати. Але ці методи часто не задовольняють за якістю результатів. Але тут маються проблеми з російською мовою, через широку морфологію.

Класичними підходами в статистичній обробці мови можна вважати використання метрики TF-IDF і її модифікацій (для виділення ключових слів), і аналіз колокацій (для виділення словосполучень).

TF-IDF - статистична міра, яка потрібна для надання оцінки важливості слова. Важлива особливість використання TF-IDF – це набір даних не має мінятись під час обчислення. Це ускладнює розрахунок, якщо треба порахувати дані в реальному часі.

Оцінка важливості слова визначається за наступними формулами:

TF (*term frequency* — частота слова):

$$tf(t,d) = n_i \Sigma n_{kk} \qquad (2.14)$$

де: n_i – число входжень слова в документ

Σn_{kk} – загальна кількість слів у документі

IDF (inverse document frequency —обернена частота документа)

$$idf(t,D)=\log_{|D||d_i \ni t_i|} \quad (2.15)$$

де: $|D|$ – кількість документів в корпусі

$|d_i \ni t_i|$ – кількість документів, у яких зустрічається термін t_i, тобто $n_i \neq 0$.

Таким чином міра TF-IDF є добутком двох попередніх оцінок:

$$tf-idf(t,d,D)=tf(t,d)\times idf(t,D) \quad (2.16)$$

Є й інші статистичні підходи для виділення термінологічних сполучень. Наприклад, один з варіантів полягає в знаходження n-слівних поєднань по заданих частотних характеристиках.

2.7.2 Лексичні методи виділення ключових слів

Спроби створити універсальний лінгвістичний метод виділення ключових слів були безуспішними. Деякі методи можна класифікувати за деякими лінгвістичним напрямками.

Лінгвістичні методи базуються на значеннях слів, використовують семантичні дані і онтології про слово. На жаль, ці методи занадто великі і тяжкі на початкових етапах: розробка онтологій є дуже затратним процесом. До того ж, ручні операції лінгвістичного аналізу текстів, є джерелом великої кількості неточностей і помилок та роблять сам процес аналізу тяжким і довгим. Тому обов'язковою умовою є присутність доступних програмних засобів, що дозволяють автоматизувати процес аналізу текстів.

Прикладом лінгвістичних методів вилучення ключових слів є методика аналізу текстів документів, що є основою автоматизованої інформаційної системи аудиту нормативних документів організації, та включає два етапи: дослідний і аналітичний. На дослідному етапі відбувається виділення ключових слів з тексту з використанням словників та операцій над множинами. Отримані ключові слова є вихідними даними для наступного

етапу, метою якого є формулювання рекомендацій щодо оптимізації електронних документопотоків і бізнес-процесів.

Існує також метод автоматичного вилучення ключових термінів з текстових документів, зав’язані на міру семантичної близькості термінів, обчисленої з використанням бази даних Вікіпедії, побудові семантичного графа. Одним з плюсів цього методу, є відсутність потреби в попередньому навчанні, тому що працює тільки з базою даних. Оцінки ефективності показують високу точність і повноту вилучення з тексту ключових термінів. Ці методи, засновані на використання значень слів, онтологій, енциклопедій, словників, в тому числі, Вікіпедії

Графові лінгвістичні методи мають великий інтерес у сфері обробки природної мови, завдяки своїй універсальності і ефективності. У них основною процедурою є побудова семантичного графа. Це граф, вершинами якого є терміни документа, ребра між двома вершинами свідчать, що терміни семантично пов'язані між собою, вага ребра є чисельним значенням семантичної близькості термінів. Ключові слова відбираються алгоритмами обробки графа. Графові методи розділяються способами відбору: 1) безлічі термінів 2) визначення близькості окремих термінів, що засновані на статистичних параметрах, і на морфологічному, синтаксичному, семантичному аналізі.

2.7.3 Гібридні методи виділення ключових слів

У попередніх підрозділах розглянуто виділення ключових слів, засноване на статистичних або лінгвістичних методах. У кожного з цих методів є свої мінуси, і лише при їх дуеті досягається найбільша точність витягнення.

Однак великим потенціалом володіють гібридні методики, в яких статистичні методи обробки доповнюються лінгвістичними процедурами і лінгвістичними базами знань. Ці методи автоматичного вилучення двуслівних

термінів з тексту на основі статистики виявлення і морфологічних шаблонів. Доведено, що використання сукупності ознак словосполучень набагато поліпшує витяг термінів.

Для поліпшення вилучення термінів і отримання більш правильних результатів (зменшення інформаційного шуму), потрібно виконати наступні умови: проведені лінгвістично орієнтовані дослідження семантичних зв'язків, термінів; системи повинні навчитися з'єднувати статистичні та лінгвістичні методи.

Тому фактично цей підхід є комбінованим, так як об'єднує статистичний і лінгвістичний методи.

2.8 Лінгвістична розмітка

Одним з методів візуалізації лінгвістичних даних є лінгвістична розмітка (рис. 2.3).

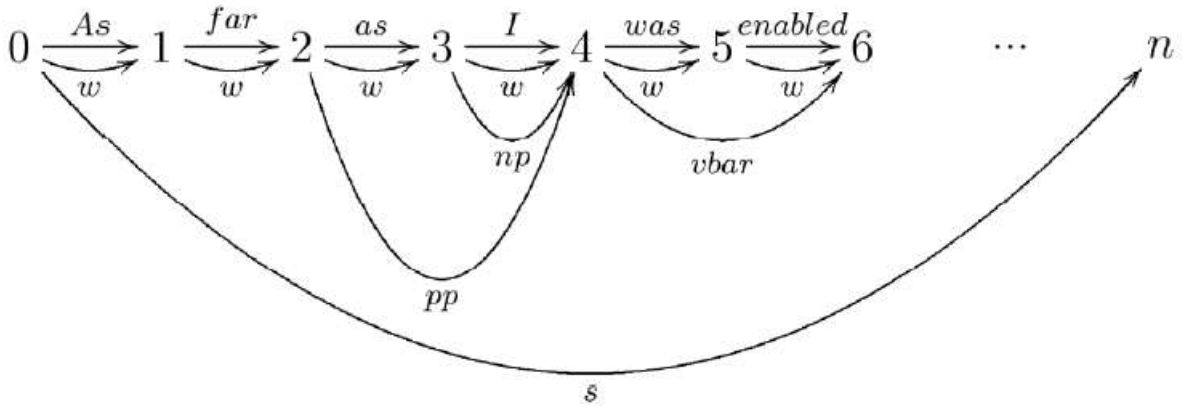


Рисунок 2.3

<s id="6293">
<wc="w" pos="IN" id="624">As</w>
<wc="w" pos="RB" id="625">far</w>
<pp id="21236">
<wc="w" pos="IN" id="626" head="yes">as</w>
<np number="singular" person="1" id="627">
<wc="w" pos="PRP" head="yes" id="628">I</w>

</np>
</pp>
<vbar voice="passive" time="past" id="629" args="+6302">
<wc="w" pos="VBD" stem="be" head="yes" id="630">was</w>
<wc="w" pos="VBN" stem="enable" id="631">enabled</w>
</vbar>

Використання спеціальної розмітки для візуалізації лінгвістичної інформації в документі виглядають зручними для обробки природної мови. Вони дозволяють легко аналізувати результати обробки користувачем або розробником. Але головна проблема підходу - представлення складних і пересічних схем. Наприклад, такі структури можуть з’явитись внаслідок неоднозначності аналізу тексту на етапі обробки. Подання таких структур значно ускладнює використовувану схему розмітки, що зводить нанівець переваги від використання стандартних програм і аналізований людиною.

ВИСНОВКИ ДО РОЗДІЛУ 2

У даному розділі були теоретично розглянуті методи, етапи та алгоритми інтелектуального аналізу текстової інформації. Сформульовано такі етапи: попередня обробка, виділення ключових слів та класифікація результатів. Розглянуто наступні методи попередної обробки: словниковий, статистичний та побудований на основі Y-інтерпретації закону Бредфорда. Для виділення ключових слів проаналізовано основні статистичні, лексичні та гібридні методи.

Для подальшого аналізу обрано наступні методи: словниковий, статистичний та побудований на основі Y-інтерпретації закону Бредфорда, TF-IDF міра, F-міра та метод лакричних шаблонів.

3 АНАЛІЗ МАТЕМАТИЧНОГО ТА АЛГОРИТМІЧНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМ КЛАСТЕРНОГО АНАЛІЗУ

3.1 Аналіз математичних основ методів кластерного аналізу

Кластерний аналіз групує об’єкти даних базуючись тільки на інформації, знайденій в даних, що описують об’єкти та їх взаємозв'язки. Мета в тому, щоб об’єкти в групі були схожими один до одного, але відрізнялися від об’єктів інших груп. Чим більша схожість об’єктів в групі, тим більше відрізняються групи, тим краща кластеризація.

В багатьох випадках поняття кластера не визначене точно. Для кращого розуміння проблеми визначення складу кластера, розглянемо рисунок 3.1, який показує 20 точок а три різні шляхи розподілу їх в кластери.

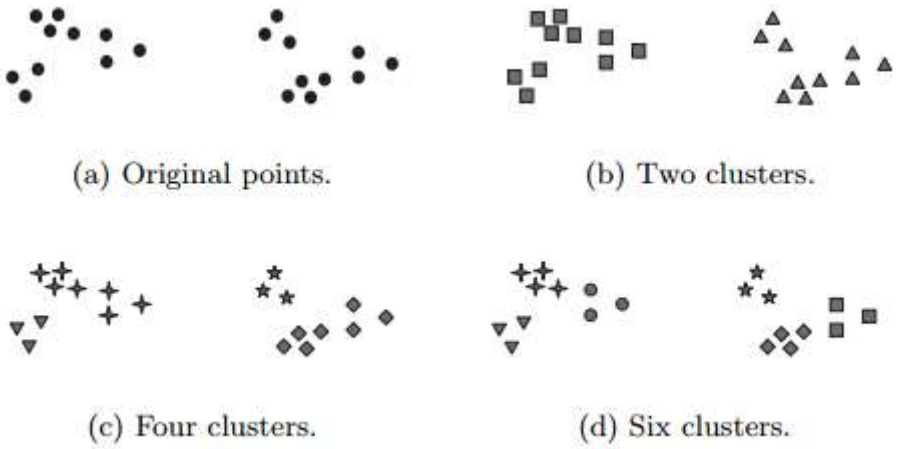


Рисунок 3.1 - Варіанти кластеризації одного набору точок

Кластерний аналіз споріднений з іншими техніками, які використовуються для розподілу об’єктів даних на групи. Наприклад, кластеризація може вважатися формою класифікації, яка здійснює маркування об’єктів міткою класу (кластеру). Мітка визначається тільки з даних. На противагу, класифікація - контрольована класифікація, тобто нові немарковані об’єкти отримують мітку класу використовуючи модель розвитку з об’єктів з

відомими мітками класів. Через це кластерний аналіз - неконтрольована класифікація.

Існують різні типи кластеризації. Розглянемо основні з них:

- Ієрархічна і неієрархічна;
- Перетинаюча, непертинаюча і нечітка;
- Повна і часткова.

Найбільше обговорюється відмінність між різними типами кластиризації, коли набори кластерів є вкладеними чи навпаки, або в більш традиційній термінології ієрархічні або не ієрархічні. Неієрархічна кластеризація – це простий поділ множини об’єктів даних на підмножини (кластери), що не перетинаються, так як кожен об’єкт даних належить тільки одній підмножині. Кожна окремо взята колекція кластерів з рисунку 3.1 неієрархічна. Якщо ми дозволяємо кластерам мати підкластери, тоді отримуємо ієрархічну кластеризацію, де набори сусідніх кластерів організовуються в дерево. Кожен вузол (кластер) в дереві (крім листових вузлів) – це об’єднання його нащадків (підкластерів) і корінь дерева містить всі об’єкти. Часто, але не завжди, листки дерева – це одинарні кластери індивідуальних об’єктів даних. Якщо ми дозволяємо кластерам бути вкладеними, то інтерпретувавши рисунок 3.1(a), ми маємо два підкластери (Рисунок 3.1(b)), кожен з яких в свою чергу має 3 підкластера – (Рисунок 3.1(d)). Кластери, показані на рисунку 3.1(a-d), в описаному порядку формують ієрархічну кластеризацію з 1, 3, 4, 6 кластерами на кожному рівні. Ієрархічна кластеризація може розглядатися як послідовність неієрархчіних кластеризацій і неієрархічна кластеризація може бути отримана як будь-який член цієї послідовності, тобто розрізом ієрархічного дерева на відповідні рівні.

Кластери, показані на рисунку 3.1 не перетинаються, оскільки кожен об’єкт відноситься тільки до одного кластера. Існує багато ситуацій, в яких точка може бути розташована більше ніж в одному кластері і в таких ситуаціях краще використовувати кластеризацію, в якій кластери можуть перетинатися.

В більш загальному сенсі перетинаюча кластеризація використовується для відображення факту, що об’єкт одночасно відноситься до більш ніж однієї групи (класу). Наприклад, людина може бути одночасно студентом і робітником університету. Перетинаюча кластеризація також часто використовується, коли об’єкт знаходиться між двома або більше кластерами і може бути віднесеним до будь-якого з них.

В нечіткій кластеризації кожний об’єкт належить кожному кластеру з вагою членства в діапазоні від 0 (абсолютно не належить) до 1 (абсолютно належить). Іншими словами кластер поводиться як нечітка множина. (Математично нечітка множина – це така, до якої об’єкти належать з вагою в діапазоні від 0 до 1. В нечіткій кластеризації накладається обмеження, що кожен кластер має мати суму ваг елементів рівною 1.) Так само техніки ймовірнісної кластеризації рахують ймовірність, з якою кожна точка належить кожному кластеру і суми цих ймовірностей рівні 1. Через наявність ваги членства або ймовірності для кожного об’єкта, сума яких 1, нечітка або ймовірнісна кластеризація не створює справжні мультикласові ситуації як зі студентом-працівником. Навпаки даний підхід найбільш підходящий для уникнення колізій при віднесенні об’єктів тільки до одного кластера, коли він близький до декількох. На практиці нечітка або ймовірнісна кластеризація часто переходить в неперетинаючу кластеризацію через віднесення кожного об’єкту до того кластера, вага членства в якому найвища.

Повна кластеризація відносить кожен об’єкт в кластер. В частковій кластеризації деякі об’єкти можуть не відноситися до визначених груп. Часто об’єкти можуть відображати шум, викиди чи нецікавий фон. З іншої сторони, повна кластеризація є бажаною.

Метою кластеризації є знаходження корисних груп об’єктів, де корисність визначається цілями аналізу даних. Не дивно, що існує декілька нотацій для доведення корисності кластера на практиці. В візуальному представленні для відображення різниці між типами кластерів використовують двовимірні точки.

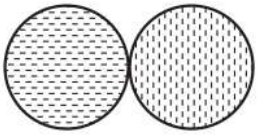
Типи кластерів:

1. Добре розділені. Кластер – це набір об’єктів, в якому кожен об’єкт схожий до інших об’єктів в кластері і не схожий до об’єктів за межами кластера. Інколи використовується поріг, щоб вказати, що всі об’єкти в кластері мають бути досить схожими до інших. Це ідеалістичне визначення кластера дійсне тільки коли кластери знаходяться далеко один від одного. Рисунок 3.2(a) показує приклад добре розділених кластерів, що складаються з двох груп точок в двовимірному просторі. Відстань між будь-якими двома точками в різних групах більша ніж відстань між будь-якими точками в одній групі. Добре розділені кластери не повинні мати форму кулі, можуть мати будь-яку.
2. Кластери, що базуються на прототипах. Кластер – набір об’єктів, в якому кожен об’єкт близький (більше схожий) до прототипу, що визначає кластер, ніж до прототипів будь-якого іншого кластера. Для даних з неперервними атрибутами прототип – це центроїд кластера, тобто середнє всіх точок в кластері. Коли центроїд не виразний, тобто дані мають категоріальні атрибути, то прототип - це меоїд, тобто точка, яка найбільш точно відображає представників кластера. Для багатьох типів даних прототипом може вважатися найбільш центральна точка і такі сутності, що належать до прототипних кластерів, називаються центроїдними кластерами. Не дивно, що такі кластери мають форму кулі. Рисунок 3.2(b) показує приклад кластера, що базується на прототипі.
3. Кластери, що базуються на графах. Якщо дані відображаються як графи, де вузли є об’єктами і ребра відображають зв’язки між об’єктами, то кластер може визначатися як зв’язна компонента, точніше група об’єктів, що зв’язані один з одним, але які не мають зав'язків з об’єктами поза групою. Важливий приклад кластерів, що базується на графах – суміжні кластери, де два об’єкта з’єднані тільки якщо вони знаходяться на певній відстані один від одного. Це означає, що кожен об’єкт в

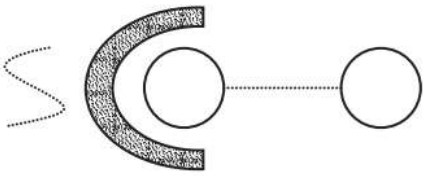
суміжному кластері ближчий до якогось іншого об’єкта в кластері ніж до точки в іншому кластері. Рисунок 3.2(c) показує приклад таких кластерів для точок в двовимірному просторі. Це визначення кластера корисне, коли кластери нерегулярні або скручені, але можуть бути проблеми, коли присутній шум, як показано на Рисунку 3.2(c) два сферичних кластери і невеликий міст з точок, що з’єднує два кластери. Інші типи кластерів, що базуються на графах також можливі.

4. Кластери, що базуються на щільності. Кластер - район згущення об’єктів, який оточений меншою щільністю об’єктів. Рисунок 3.2(d) відображає кластери, що базуються на щільності для даних з рисунку 3.2(c) з доданим шумом. Два кулясті кластера не об’єднані, як на рисунку 3.2(c), бо міст між ними злився з шумом. Також крива, що представлена на рисунку 3.2(c) зникла через шум і не формує кластер на рисунку 3.2(d). Визначення, що базується на щільності, зазвичай використовується в зашумленому середовищі з можливими викидами. На противагу кластер, що базується на графах не буде працювати добре на даних з рисунку 3.2(d), бо шум зруйнує міст між кластерами.
5. Кластери з спільними ознаками. Більш загально ми можемо визначити кластер як набір об’єктів з деякими спільними ознаками. Це визначення об’єднує всі попередні визначення кластера, об’єкти в центроїдному кластері мають спільні ознаки, за якими вони є близькими до центроїд чи медоїд. Цей підхід також включає нові типи кластерів. Розглянемо кластери зображені на рисунку 3.2 (e). Трикутна площа (кластер) суміжна до прямокутної і є два сплетені кулясті кластери. В обох випадках алгоритм кластеризації потребує специфічних умов для успішного виявлення цих кластерів. Процес знаходження цих кластерів називається концептуальною кластеризацією.

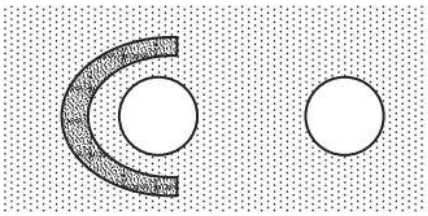
(a) Well-separated clusters. Each point is closer to all of the points in its cluster than to any point in another cluster.



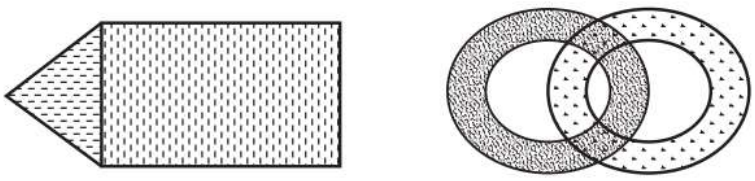
(b) Center-based clusters. Each point is closer to the center of its cluster than to the center of any other cluster.



(c) Contiguity-based clusters. Each point is closer to at least one point in its cluster than to any point in another cluster.



(d) Density-based clusters. Clusters are regions of high density separated by regions of low density.



(e) Conceptual clusters. Points in a cluster share some general property that derives from the entire set of points. (Points in the intersection of the circles belong to both.)

Рисунок 3.2 – Типи кластерів

3.2 Алгоритмічна реалізація методів кластерного аналізу

Алгоритми кластеризації можна класифікувати наступним чином.

Методи за способом опрацювання даних:

Ієрархічні методи:

- Агломеративні методи AGNES (Agglomerative Nesting):
  - CURE;
  - ROCK;
  - CHAMELEON і т.д.
- Розділяючі методи DIANA (Divisive Analysis):
  - BIRCH;
  - MST і т.д.

Неієрархічні методи:

- Ітеративні:
  - K-means;
  - PAM (k-means + k-medoids);
  - CLOPE;
  - LargeItem і т.д.

Методи за способом аналізу даних:

- Чіткі;
- Нечіткі.

Методи за кількістю застосувань алгоритмів кластеризації:

- З одноетапною кластеризацією;
- З багатоетапною кластеризацією.

Методи за можливістю розширення обсягу опрацьовуваних даних:

- Масштабовані;
- Немасштабовані.

Методи за часом виконання кластеризації:

- Потокові (on-line);
- Непотокові (off-line).

В даному розділі буде розглянуто три прості, але важливі техніки для представлення багатьох концепцій в кластерному аналізі:

- K-середніх. Це неієрархічна кластеризація, що базується на прототипах, що намагається знайти специфічну групу кластерів (K), які відображаються через центроїди.
- Агломеративна ієрархічна кластеризація. Цей підхід кластеризації відноситься до колекції технік кластеризації, що виконують ієрархічну кластеризацію починаючи з кожної точки як окремого кластера і далі замінюючи одинарний кластер кластером з двох найближчих точок замість однієї. Деякі з цих технік мають інтерпретацію в термінах кластеризації, що базуються на графах,

коли інші мають інтерпретацію в термінах кластеризації, що базуються на прототипах.

- DBSCAN. Даний алгоритм кластеризації базується на щільності розподілу і виконує неієрархічну кластеризацію, в якій кількість кластерів автоматично визначається алгоритмом. Регіони з малою густиною точок класифікуються як шум і не розглядаються. DBSCAN не виконує повну кластеризацію.

3.2.1 К-середніх

Розглянемо алгоритм К-середніх. Дана техніка кластеризації базується на прототипах і створює однорівневий розподіл об'єктів даних. Існує безліч підходів, але два найвідоміших: К-середніх і К-медоїд. К-середніх визначає прототип в термінах центроїда, який зазвичай середній серед групи точок і зазвичай застосовується до об'єктів в n-вимірному просторі. К-медоїд визначає прототип в термінах медоїда, який є найхарактернішим представником групи точок і може застосовуватися до широкого спектру даних. В той час як центроїд не відповідає жодній точці даних, медоїд, за визначенням, має бути точкою даних з набору. Далі буде розглядатися К-середніх алгоритм, що є найбільш широко використовуваним алгоритмом.

К-середніх техніка кластеризації проста і необхідно почати її опис з опису основного алгоритму. Спочатку вибирається К початкових центроїдів, де К – параметр, що визначається користувачем і означає кількість бажаних кластерів. Кожна точка далі призначається найближчому центроїду і кожна колекція точок призначена центроїду – кластер. Центроїд кожного кластера далі оновлюється, базуючись на точках в кластері. Далі повторюється процес присвоєння і оновлення доки точки перестануть змінювати кластери або центроїди будуть залишатися тими ж самими.

Формалізуємо алгоритм К-середніх.

1. Вибрати К точок як початкові центроїди.

2. Повторити поки центроїди перестануть змінюватися:
    - Сформувати К кластерів через призначення кожної точки найближчому центроїду.
    - Перерахувати центроїд кожного кластера.\

Реалізація К-середніх зображена на рисунку 3.3, який показує як починаючи з трьох центроїд остаточні кластери знайдені за 4 кроки призначення-оновлення.

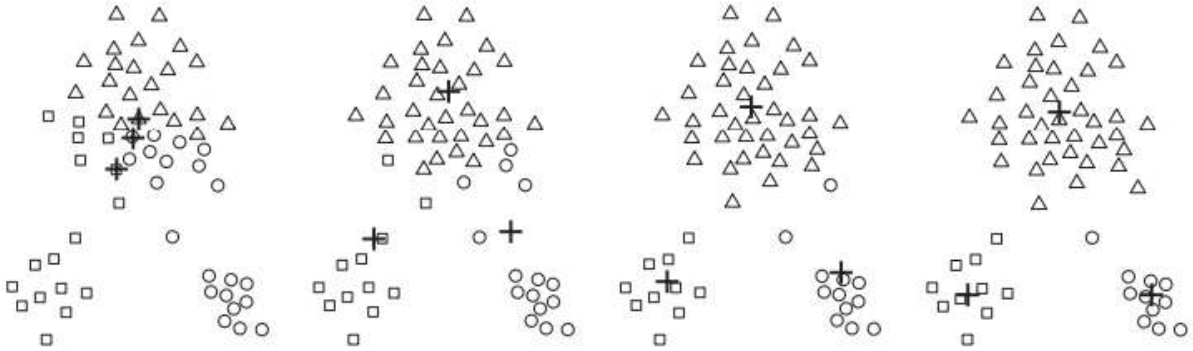


Рисунок 3.3 – Реалізація алгоритму К-середніх

На цьому рисунку показана кластеризація методом К-середніх, де кожний компонент рисунку показує центроїди на початку ітерації і призначення точок цьому центроїду. Центроїди позначені символом "+", всі точки, що належать одному кластеру мають ту ж саму фігуру позначення.

На першому кроці, показаному на Рисунку 3.3(a), точки призначаються початковим центроїдам, які знаходяться в найбільшій групі точок. В даному випадку як центроїд береться середнє значення. Після того як точки призначені центроїдам, центроїди оновлюються. Рисунок на кожному кроці показує центроїди на початку ітерації і призначення точок цим центроїдам. На другому кроці точки призначаються уже оновленим центроїдам і центроїди оновлюються знову. На кроках 2,3,4, які показані на рисунку 3.3(b), (c), (d), центроїди переміщуються до маленьких груп точок внизу рисунку. Коли алгоритм К-середніх припиняє роботу, оскільки змін не відбувається, центроїди опиняються в місцях групування точок.

Для деяких комбінацій функцій близькості і типів центроїдів К-середніх завжди має рішення, тобто К-середніх знаходить стан, в якому точки не переміщуються з одного кластера в інший, а отже центроїди не змінюються.

Розглянемо кожен крок алгоритму детально і потім проведемо аналіз складності алгоритму і пам'яті, яку він потребує.

Для призначення точки найближчому центроїду потрібно виміряти близькість розташування. Евклідова відстань ($L_{2)}$ часто використовується для точок даних в евклідовому просторі, в той же час косинус схожості більш підходить для документів. Можу бути декілька типів вимірювання відстані, для даних типів даних.

Різновиди функцій відстані:

1. Евклідова відстань. Найбільш поширена функція відстані. Являє собою геометричну відстань в багатомірному просторі.

$$\rho(x, x') = \sqrt{\sum_{i}^{n} (x_i - x_i^{'})^2}$$

Евклідова відстань (і її квадрат) обчислюється по вхідним, а не по стандартизованим даним. Це звичайний спосіб її обчислення, який має певні переваги (наприклад, відстань між двома об'єктами не зміниться при введенні нового об'єкту, що може бути викидом). Тим не менш, на відстань можуть сильно впливати відмінності між осями, по координатах яких обчислюється дана відстань. Наприклад, якщо одна із осей виміряна в сантиметрах, а потім ви переведете її в міліметри (помноживши на 10), то вихідна евклідова відстань, обчислена за координатами, сильно зміниться, і результати кластерного аналізу можуть сильно відрізнятися від попередніх.

2. Квадрат евклідової відстані. Для того щоб надати більшої ваги найбільш віддаленим один від одного об'єктам можемо використати квадрат евклідової відстані шляхом піднесення до квадрату стандартної евклідової відстані.

$$\rho(x,x') = \sum_{i}^{n} (x_i - x_i')^2$$

3. Манхеттенська відстань (відстань міських кварталів). Ця відтань розраховується як середнє різниць по координатам. В більшості випадків ця міра відстані приводить до результатів, схожим до відстані Евкліда. Однак, для цієї міри вплив окремих викидів менший ніж при використанні евклідової відстані, оскільки в даній ситуації координати не підносяться до квадрату.

$$\rho(x,x') = \sum_{i}^{n} | x_i - x_i' |$$

4. Відстань Чебишева. Дана метрика використовується, коли необхідно визначити два об'єкта як відмінні, якщо вони відрізняються по якомусь одному виміру.

$$\rho(x,x') = \max(| x_i - x_i' |)$$

5. Степенева відстань. Застосовується в випадку, коли необхідно збільшити чи зменшити вагу, що відноситься до розмірності, для якої існуючі об'єкти сильно відрізняються.

$$\rho(x,x') = \sqrt[r]{\sum_{i}^{n} (x_i - x_i')^p}$$

де r і p - параметри, що визначаються користувачем. Параметр p відповідає за поступове зважування різниць по окремим координатам, параметр r відповідає за прогресивне зважування великих відстаней між об'єктами. Якщо обидва параметра рівні двом, то дана відстань співпадає з відстанню Евкліда.

6. Відсоток неузгодженості. Дана метрика використовується в тих випадках, коли дані категоріальні.

$$P = VALUE \left| A_i \neq B_i \right|$$

Зазвичай вимірювання близькості в алгоритмі К-середніх просте, оскільки алгоритм повторно розраховує близькість кожної точки до кожного центроїда. В деяких випадках, таких як маловиміний евклідовий простір,

можливим стає уникати безлічі близькостей, пришвидшуючи алгоритм К-середніх.

Таблиця 3.1 – Позначення змінних для формалізації алгоритму

Символ	**Опис**
x	Об’єкт
C_i	і-й кластер
c_i	Центроїд кластера C_i
c	Центроїд всіх точок
m_i	Кількість точок в і-му кластері
m	Кількість об’єктів в наборі
K	Кількість кластерів

Крок 4 алгоритму К-середніх починається з перерахувнку центроїдів всіх кластерів, центроїди можуть варіюватися в залежності від функції близькості та мети кластеризації. Мета кластеризації зазвичай виражається цільовою функцією, що залежить від близькості точок одна до одної чи до центроїдів, наприклад мінімізований квадрат відстані кожної точки до найближчого центроїда. Ключовою точкою є те, що якщо ми визначили функцію близькості та цільову функцію, то центроїд, який ми вибираємо часто визначається математично.

Розглянемо дані, близькість яких визначається в евклідовому просторі. Для нашої цільової функції, яка вимірює якість кластеризації, використовуємо суму квадратів помилок (СКП), яка також відома як розсіювання. Іншими словами, ми рахуємо помилку кожної точки даних, тобто евклідову відстань до найближчого центроїда і далі загальну суму квадратів помилок. Дано два різні набори кластерів, які створені двома запусками алгоритму К-середніх, ми надаємо перевагу одному з найменшими квадратом помилки, оскільки це означає, що центроїди цієї кластеризації краще показують характерні точки в кластері.

Використовуючи позначення в таблиці 3.1, визначимо СКП формально:

$$\text{СКП} = \sum_{i=1}^{K} \sum_{x \epsilon Ci} dist(c_i, x)^2$$

,де dist - стандартна евклідова відстань між двома об’єктами в евклідовому просторі.

Центроїди, які мінімізують СКП кластера – середнє. Використовуючи позначення в таблиці 3.1, центроїд (середнє) і-го кластера визначається виразом:

$$\mathbf{c}_i = \frac{1}{m_i} \sum_{\mathbf{x} \in C_i} \mathbf{x}$$

Центроїд кластера з трьох точок двовимірного простору (1,1), (2,3), (6,2) - ((1+2+6)/3, (1+3+2)/3) = (3,2).

Кроки 3 та 4 алгоритму К-середніх є безпосередньою спробою мінімізувати СКП. Крок 3 формує кластери через присвоєння точок їх найближчим центроїдам, які мінімізують СКП для даного набору центроїдів. Крок 4 перераховує центроїди для подальшої мінімізації СКП. Проте, дії алгоритму К-середніх в кроках 3 та 4 тільки гарантують знаходження локальних мінімумів в відношенні до СКП так як базуються на оптимізації СКП для специфічних виборів центроїдів і кластерів, а не для всіх можливих виборів.

Для ілюстрації того, що алгоритм К-середніх не обмежується даними в евклідовому просторі, розглянемо кластеризації документів і косинусну міру схожості. Припустимо документи відображаються через матрицю термінів. Наша мета максимізувати схожість документів в кластері до центроїда кластера, ця величина називається зв’язністю кластера. Аналогом загальної СКП є загальна зв’язність , яка виражається виразом.

$$\text{Total Cohesion} = \sum_{i=1}^{K} \sum_{\mathbf{x} \in C_i} cosine(\mathbf{x}, \mathbf{c}_i)$$

Існує безліч варіантів вибору функції близькості, кластерів, цільової функції, які можуть бути використані в базовому алгоритмі К-середніх і які гарантують результат. Таблиця 3.2 показує деякі можливі способи, включаючи два вище обговорені.

Таблиця 3.2 – Типові вибори функції близькості, центроїдів та цільової функції

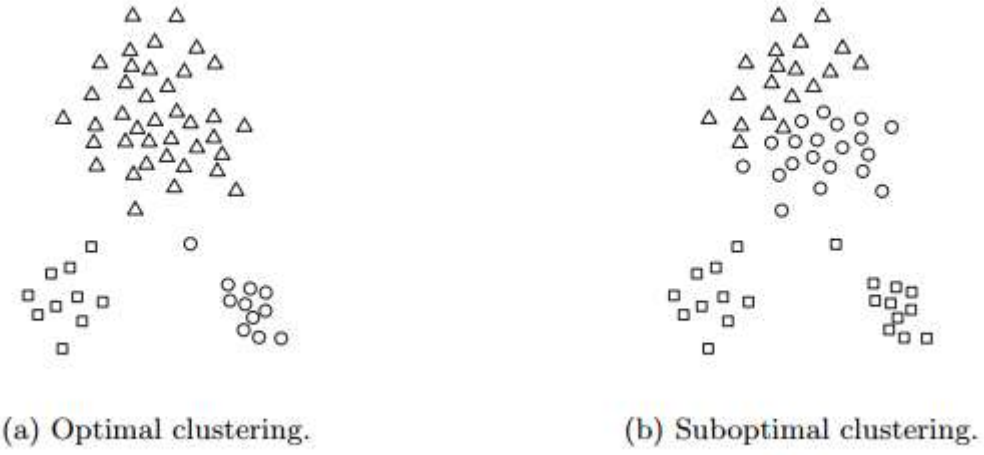
Функція близькості	Центроїд	Цільова функція
Манхеттенська відстань (L1)	Медіана	Мінімізувати суму L1 відстані об’єкта до центроїда кластера
Квадрат евклідової відстані (L^2_2)	Середнє	Мінімізувати суму квадрату L2 відстані об’єкта до центроїда кластера
Косинус	Середнє	Максимізувати суму косинусної схожості об’єкта до центроїда кластера
Розбіжність Бергмана	Середнє	Максимізувати суму розбіжності Бергмана об’єкта до центроїда кластера

Розбіжність Бергмана – клас вимірювання близькості, який включає квадрат евклідової відстані, відстань Махаланобіса, і косинусну близькістсь. Важливість розбіжності Бергмана в тому, що така функція може бути використана як базис К-середніх кластеризації з середнім як центроїдом. Конкретно, якщо використовується розбіжність Бергмана, як функція близькості, то результат алгоритму кластеризації має звичайні властивості К-середніх в виношенні сходження, локальних мінімумів і т. д. Більш того, властивості такого алгоритму кластеризації можуть бути розвинені для всіх можливих розбіжностей Бергмана. Дійсно, алгоритм К-середніх, який використовує косинусну близькість або квадрат евклідової відстані, - конкретний екземпляр загальноо алгоритму кластеризації, що базується на розбіжності Бергмана.

Для подальшого розгляду ми будемо використовувати двовимірні дані, оскільки на них легше пояснити К-середніх та його властивості для цього типу даних. Але алгоритм К-середніх можу бути використаний для широкого сперктру типів даних, таких як документи або часові ряди.

Коли використовується випадкова ініціалізація центроїдів, то різні запуски алгоритму К-середніх дадуть різні значення загальної СКП. Ми зобразили це за допомогою двовимірних точок на рисунку 3.4, який містить 3

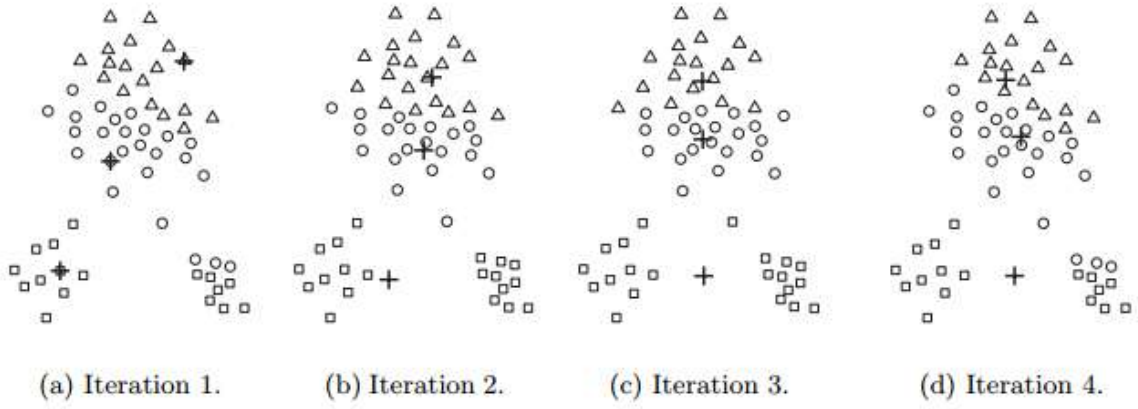
кластери точок. Рисунок 3.4(a) показує рішення кластеризації, глобальний мінімум СКП для трьох кластерів, в той же час рисунок 3.4(b) показує неоптимальну кластеризацію, тобто тільки локальний мінімум.



(a) Optimal clustering. (b) Suboptimal clustering.

Рисунок 3.4 – Оптимальні та неоптимальні кластери

Випадково вибрані початкові центроїди можуть створювати невдалі результати. Рисунки 3.4 та 3.5 показують кластери, що є результатом двох окремих випадків вибору початкових центроїдів. (Для обох рисунків позиція центроїда кластера на різних ітераціях зображується двома перетнутими лініями.) На Рисунку 3.4, навіть незважаючи на те, що початкові центроїди знаходяться в одному кластері, мінімальне СКП все ж знаходиться. На рисунку 3.5 незважаючи на те, що початкові кластери розподілені краще, отримуємо неоптимальну кластеризацію, з більшою квадратною помилкою.



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3. (d) Iteration 4.

Рисунок 3.5 – невдалі початкові центроїди для К-середніх

Приклад 2 (Межі випадкової ініціалізації). Техніка, що зазвичай використовується для вирішення проблеми вибору початкових центроїдів, - виконання декількох запусків, кожен з різним набором випадково вибраних початкових центроїдів і потім вибір групи кластерів з мінімальним СКП. Не зважаючи на простоту, ця стратегія може працювати не дуже добре, залежачи від наборів даних та кількості кластерів, що знаходяться. Продемонструємо це на простому наборі даних, зображеному на Рисунку 6 (а). Дані складаються з двох пар кластерів, де кластери в кожній парі (пари горизонтальні) ближчі один до одного, ніж до кластерів в іншій парі.

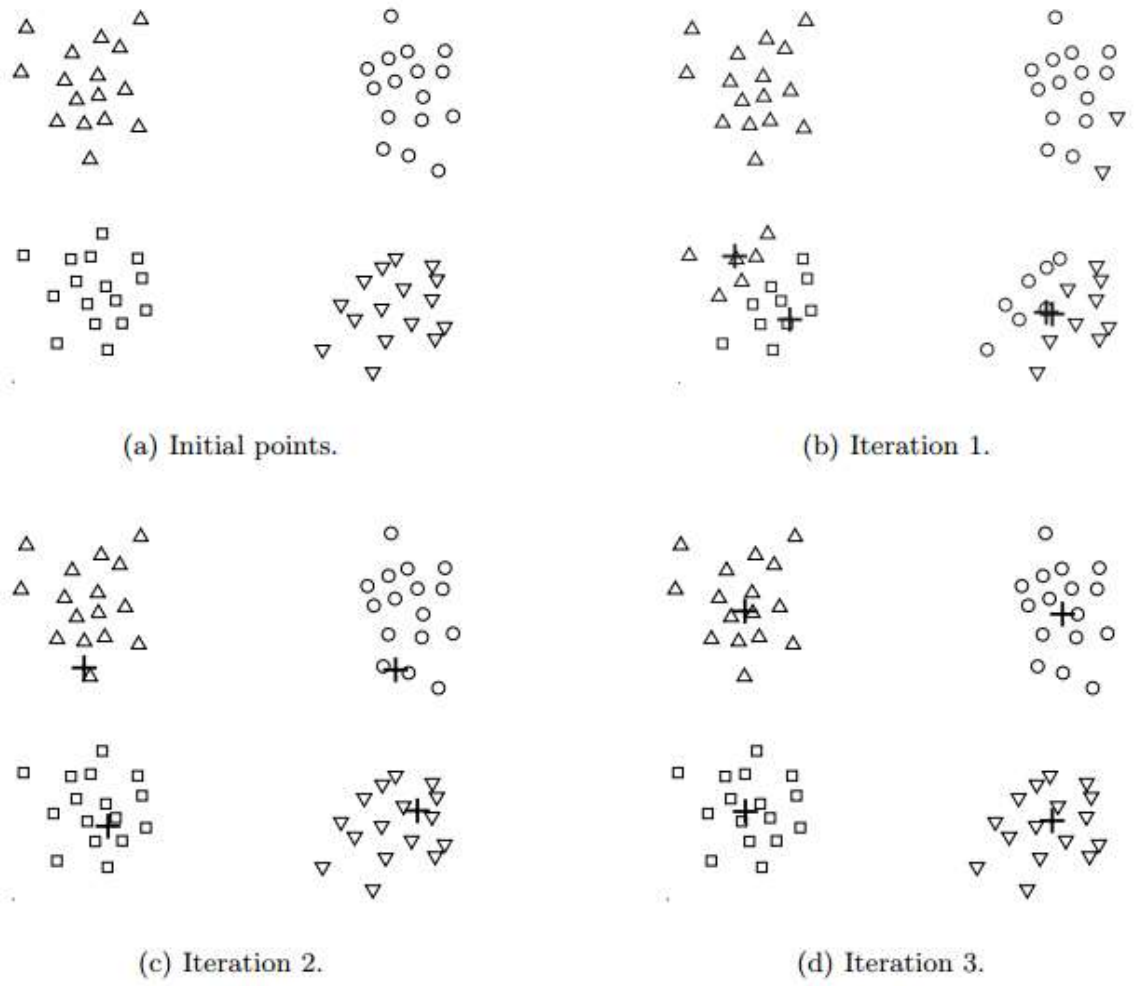


Рисунок 3.6 – Дві пари кластерів з початковими

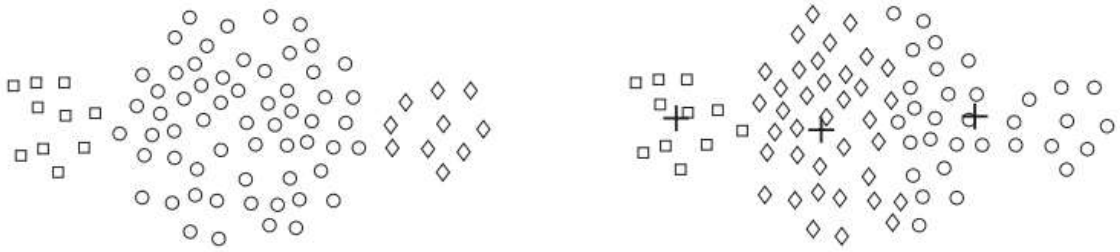
Вимоги до пам’яті для алгоритму К-середніх помірні, тому що тільки точки даних і центроїди зберігаються. Конкретно потребує пам’яті O((m+K)n), де m - кількість точок і n - кількість атрибутів. Часові вимоги для алгоритму К-середніх також помірні, зазвичай лінійна складність по кількості точок.

Зокрема потребує часу O(I*K*m*n), де I - кількість ітерацій необхідних для збіжності. Як зазначалося, I часто мале і зазвичай може бути обмеженим так як більшість змін відбуваються на декількох початкових ітераціях. Отже, алгоритм К-середніх лінійний по m, кількість точок, і ефективний як і простий, де К - кількість кластерів, що менше m.

Одна з проблем, пов'язаних з базовим алгоритмом К-середніх, - це те, що порожні кластери можуть бути отримані якщо ніякі точки не будуть виділені в кластер на етапі присвоєння. Якщо таке трапляється, то необхідно замінити центроїди, оскільки в протилежному випадку квадрат помилки буде більшим ніж потрібно.

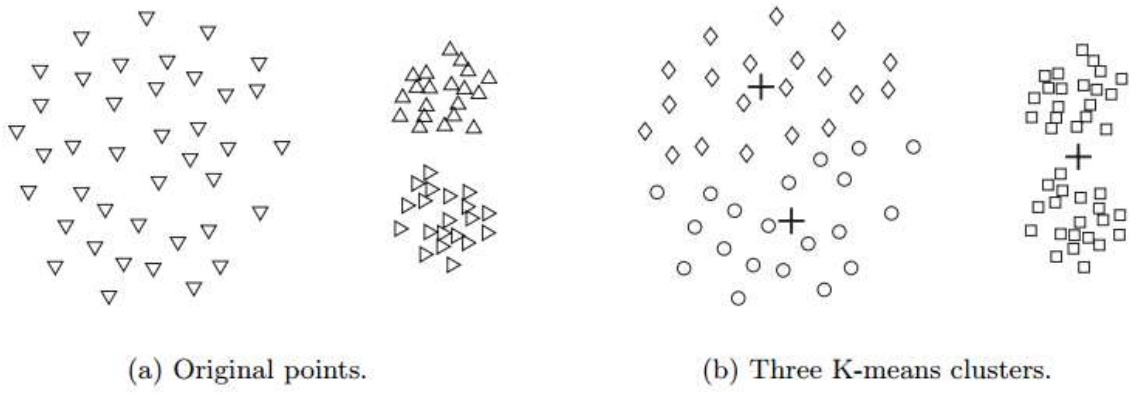
Коли використовується критерій квадрату помилки, викиди можуть надмірно впливати на знайдені кластери. Зокрема, коли присутні викиди, результуючі центроїди кластерів можуть не відображати в достатній мірі елементи кластера і СКП буде збільшеною. Через це часто корисно знайти викиди і ліквідувати їх зарані.

К-середніх і його варіації мають певні обмеженні щодо знаходження різних типів кластерів. К-середніх має складності з знаходженням кластерів, що мають не сферичну форму чи розміри (форму), що сильно відрізняються. Це показано на Рисунках 7, 8, 9.



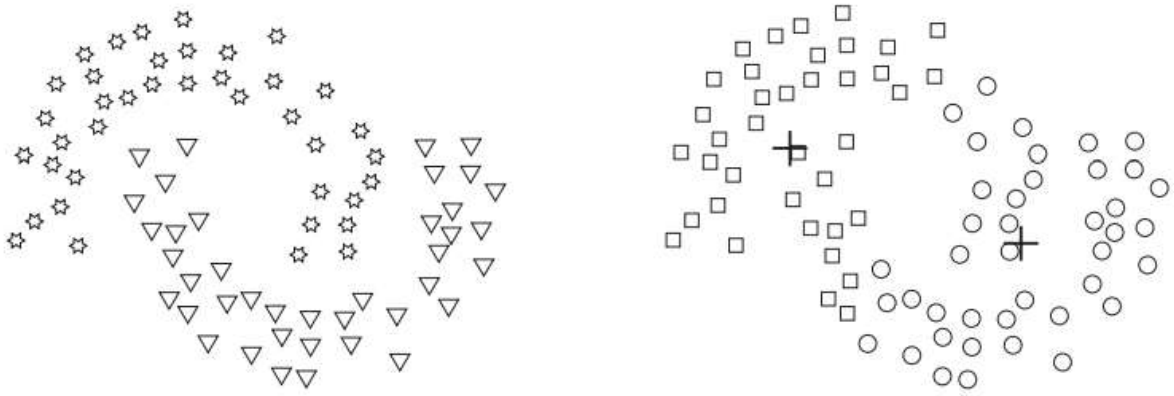
(a) Original points. (b) Three K-means clusters.

Рисунок 3.7 – К-середніх при роботі з кластерами різних розмірів



(a) Original points. (b) Three K-means clusters.

Рисунок 3.8 – К-середніх при роботі з кластерами різної густини



(a) Original points. (b) Two K-means clusters.

Рисунок 3.9 – К-середніх при роботі з кластерами несферичної форми

На рисунку 3.7 К-середніх не може знайти три кластери, тому що один з кластерів набагато більший за два інші, найбільший кластер розпадається і менші кластери комбінуються з порцією точок з найбільшого кластера. На рисунку 3.8 К-середніх помиляється при знаходженні трьох кластерів, тому що два менші кластери мають меншу густину ніж третій. На рисунку 3.9 К-середніх знаходить два кластери, що є сумішшю точок двох справжніх кластерів, тому що вони мають не сферичну форму.

К-середніх – простий і може використовуватися для різних типів даних. Він ефективний навіть не зважаючи на здійснення багаторазового запуску. Деякі варіації К-середніх навіть більш ефективні і менш сприйнятливі до проблем ініціалізації. К-середніх не підходить для всіх типів даних. Також він не може опрацьовувати не сферичні кластери, кластери різних розмірів чи густини. К-середніх також має проблеми при кластеризації даних, що містять

викиди. Знаходження та видалення викидів може допомогти в деяких випадках.

3.2.2 Агломеративна ієрархічна кластеризація

Ієрархічна кластеризація – друга важлива категорія методів кластерного аналізу. Дані підходи відносно давні в порівнянні з багатьма алгоритмами кластеризації, але вони до цих пір широко використовуються. Є дві основні категорії для узагальнення ієрархічної кластеризації:

- Агломеративна – кластеризація починається з точок як індивідуальних кластерів і на кожному кроці об’єднуються найближчі пари кластерів. Це дозволяє визначення поняття близькості кластерів.
- Розділяюча – кластеризація починається з одного кластера, що включає всі об’єкти і на кожному кроці відділяється кластер доки всі кластери не міститимуть одну точку. В цьому випадку треба вирішити який кластер розділити на кожному кроці і як здійснити цей розділ.

Ієрархічна кластеризація зазвичай зображається графічно з використанням деревоподібних діаграм, що називаються дендрограмами, які відображають взаємозв’язок кластер-підкластер в порядку того як кластери були об’єднані чи розділені.

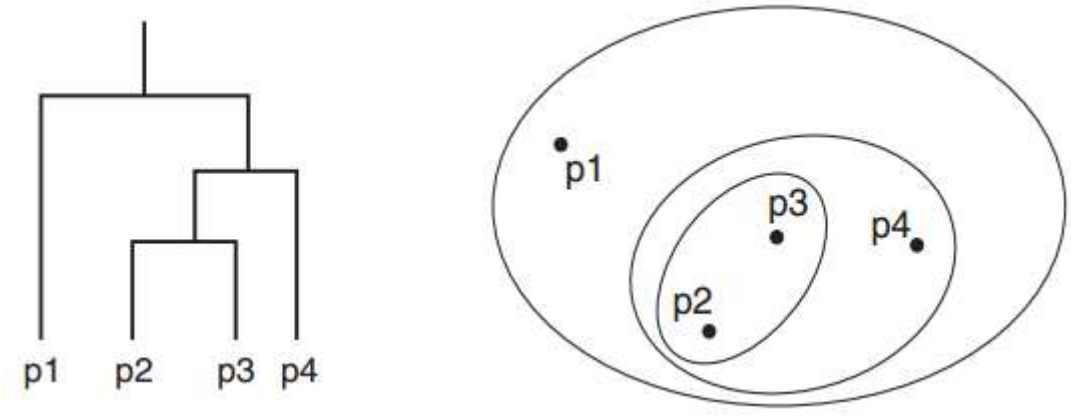


Рисунок 3.10 – Ієрархічна кластеризація чотирьох точок в вигляді дендрограми

Для набору двовимірних точок, таких як використані в прикладі, ієрархічна кластеризація може бути також зображена в вигляді вкладених кластерних діаграм. Рисунок 3.10 показує два типи діаграм для двовимірних точок.

Багато технік алгомеративної ієрархічної кластеризації є варіаціями єдиного підходу: починаючи з одноточкових кластерів, об'єднувати два найближчих кластера доки не буде отриманий єдиний кластер. Даний підхід формалізований в алгоритмі нижче:

1. Обчислити матрицю близькостей при необхідності.
2. Об'єднати два найближчі кластери.
3. Оновити матрицю близькостей для відображення близькостей між щойно утвореним кластером і кластерами, що залишилися.
4. Повторювати поки не буде отриманий один суцільний кластер.

Ключовою операцією в алгоритмі є обчислення близькості між двома кластерами і визначення цієї близькості відрізняє різні агломеративні ієрархічні техніки кластеризації. Багато агломеративних ієрархічних технік кластеризації, такі як MIN, MAX, Group Average прийшли від зображення кластерів в вигляді графів.

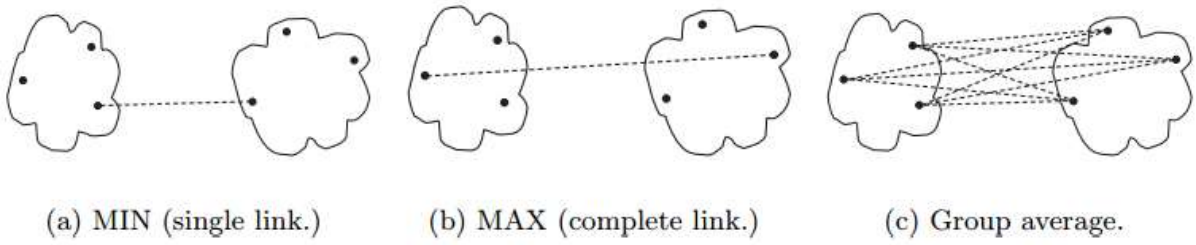


Рисунок 3.11 – Визначення близькості кластерів на основі теорії графів

MIN (метод найближчого сусіда) визначає близькість як близькість між двома точками в різних кластерах або в термінах теорії графів, як найкоротше ребро між вузлами в різних підгрупах вузлів. MAX (метод найвіддаленішого сусіда) визначає близькість як найдовше ребро між двома вузлами в різних групах точок. Інший підхід, що базується на теорії графів, - техніка group average, визначає близькість як середню довжину ребра всіх пар точок в різних кластерах. Рисунок 3.11 зображає всі ці підходи.

В прототипному підході, де кластер відображається через його центроїд, визначення близькості кластерів більш природне. Альтернативною технікою є метод Варда, де кластери також відображаються через центроїди, але вимірювання близькості кластерів виражається в термінах зростання СКП, що обчислюється для об'єднання цих кластерів. Метод Варда прагне мінімізувати суму квадратів відстаней точок до центрів центроїдів кластерів.

Базовий агломеративний ієрархічний алгоритм кластеризації представлений матрицею близькості. Це потребує пам'яті $m^2/2$ близькостей, де m - це кількість точок даних. Пам'ять, що необхідна для того, щоб слідкувати за кластерами пропорційна кількості кластерів, m-1, без врахування одиничних кластерів. Загальна складність по пам'яті рівна $O(m^2)$.

Аналіз базового агломеративного ієрархічного алгоритму кластеризації простий в відношенні обчислювальної складності. $O(m^2)$ часу необхідно для обчислення матриці близькості. Після цього кроку m-1 ітерацій включають кроки 3 та 4, тому що є m кластерів на початку і два кластери об'єднуються на кожній ітерації. Якщо здійснювати лінійний пошук по матриці близькості то для і-ї ітерації крок 3 потребує $O((m-i+1)^2)$ часу, що пропорційно квадрату кількості кластерів. Крок 4 потребує $O(m-i+1)$ часу для оновлення матриці близькості після об'єднання двох кластерів. Без модифікацій часова складність рівна $O(m^3)$. Якщо відстані від кожного кластера до всіх інших зберігаються як відсортований список (або кучу), можливо зменшити вартість знаходження двох найближчих кластерів до $O(m-i+1)$. Додаткова складність виникає при зберіганні даних у відсортованому списку чи кучі, загальний час необхідний для ієрархічної кластеризації за даним алгоритмом рівний $O(m^2 \log m)$.

Для ілюстрації поведінки різних варіацій алгоритмів ієрархічної кластеризації використаємо вибірку даних з 6 двовимірних точок, які показані на рисунку 3.12.

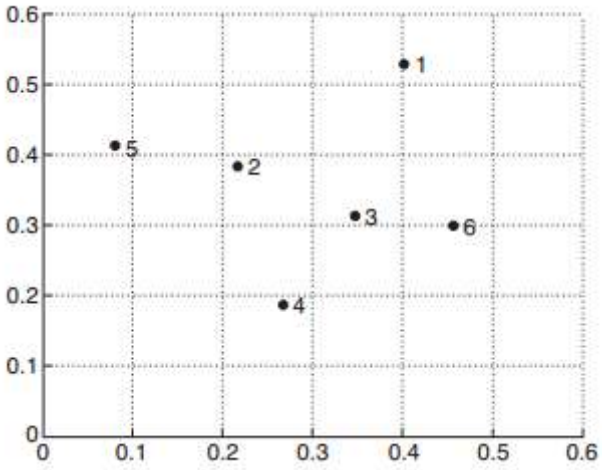


Рисунок 3.12 – Набір з шести двовимірних точок

Координати x та y і евклідові відстані між ними показані в таблиці 3.3 і 3.3.

Таблиця 3.3 – x та y координати шести точок

Point	x Coordinate	y Coordinate
p1	0.40	0.53
p2	0.22	0.38
p3	0.35	0.32
p4	0.26	0.19
p5	0.08	0.41
p6	0.45	0.30

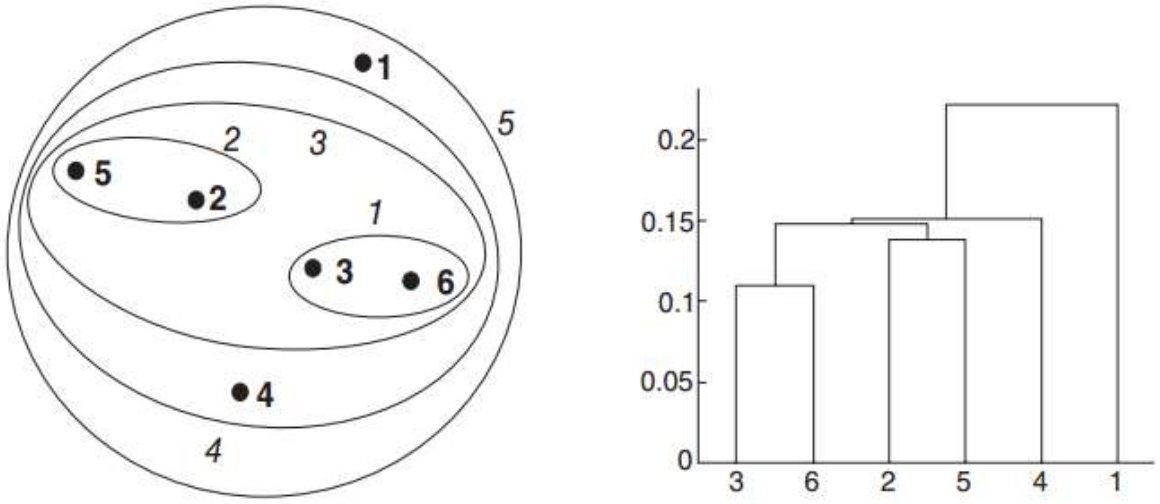
Таблиця 3.4 – Матриця евклідових відстаней для шести точок

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

MIN або Single Link. Для MIN версії ієрархічної кластеризації близькість двох кластерів визначається як мінімальна відстань між будь-якими двома точками в двох різних кластерах. В термінології теорії графів якщо почати з усіх точок як одиничних кластерів і додати зв’язки між точками, то

найкоротші зв’язки об’єднають точки в кластер. Техніка є хорошою для обробки нееліптичних фігур, але чутлива до шумів та викидів.

Рисунок 3.13 показує результат застосування техніки Single Link на прикладі 6 точок.



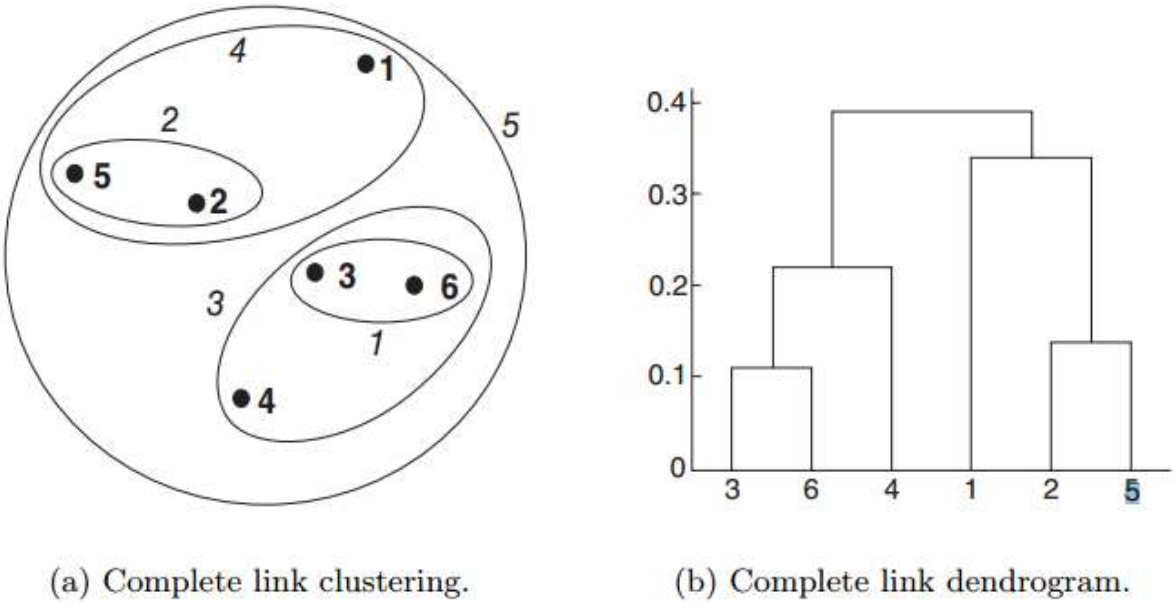
(a) Single link clustering. (b) Single link dendrogram.

Рисунок 3.13 – Кластеризація Single Link

Рисунок 3.13(a) показує вкладені кластери як послідовність вкладених еліпсів, де числа асоціюються з індикатором порядку кластеризації. Рисунок 3.13(b) показує цю ж саму інформацію в вигляді дендрограми. Висота, на якій два кластери об’єднуються на дендрограмі відображає відстань між двома кластерами. Наприклад, з таблиці 3.4 бачимо, що відстань між точками 3 та 6 рівна 0,11 і це висота, на якій точки об’єднуються в кластер на дендрограмі.

Для версії ієрархічної кластеризації Complete Lin чи MAX чи CLIQUE близькість двох кластерів визначається як максимум відстані (мінімум схожості) між будь-якими двома точками в різних кластерах. Використовуючи теорію графів, якщо починати з точки як окремого кластера і додати зв’язки між точками, спочатку найкоротші зв’язки, потім група точок не є кластером доки всі точки повністю не зв’язані. Complete Linq менш чутлива до шумів та викидів, але може розбити великі кластери і прихильна до сфероподібих фігур.

Рисунок 3.14 показує результат застосування MAX до набору з шести точок.



(a) Complete link clustering. (b) Complete link dendrogram.

Рисунок 3.14 – Complete Linq кластеризація

Точки 3 та 6 об'єднуються першими. Однак, {3,6} об'єднуються з {4}, а не {2,5} чи {1}, тому що

dist({3,6}, {4}) = max(dist(3,4), dist(6,4))
= max(0.15, 0.22)
= 0.22.

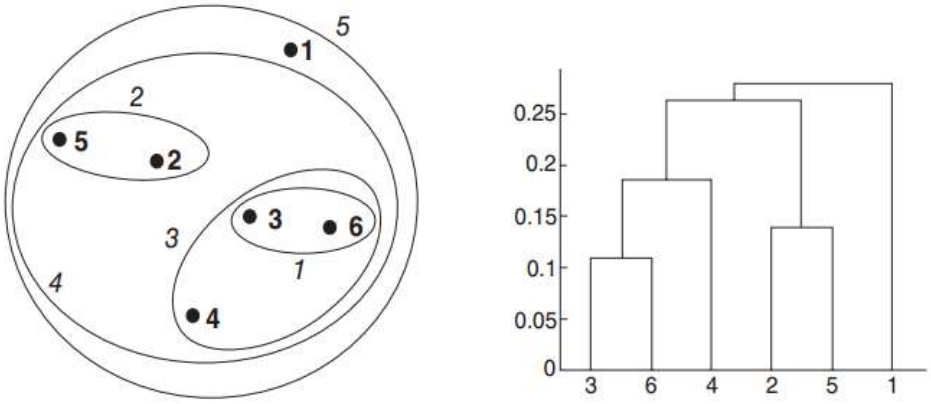
dist({3,6}, {2,5}) = max(dist(3,2), dist(6,2), dist(3,5), dist(6,5))
= max(0.15, 0.25, 0.28, 0.39)
= 0.39.

dist({3,6} ,{1}) = max(dist(3,1), dist(6,1))
= max(0.22, 0.23)
= 0.23.

Для даної версії ієрархічної кластеризації Group Average близькість двох кластерів визначається як середня попарна близькість між всіма парами точок в різних кластерах. Даний підхід є чимось середнім між Single Linq та Complete Linq. Для даного методу близькість *proximity*(C_i, Cj) кластерів C_i та C_j розміру m_i та m_j відповідно виражена наступним виразом

$$proximity(C_i, C_j) = \frac{\sum_{\substack{\mathbf{x} \in C_i \\ \mathbf{y} \in C_j}} proximity(\mathbf{x}, \mathbf{y})}{m_i * m_j}.$$

Рисунок 3.15 показує результат застосування методу Group Average.



(a) Group average clustering. (b) Group average dendrogram.

Рисунок 3.15 – Group Average кластеризація

Для демонстрації роботи методу, обчислимо відстані між деякими кластерами.

dist({3,6,5}, {1}) = (0.22 + 0.37 + 0.23)/(3*1)
= 0.28.

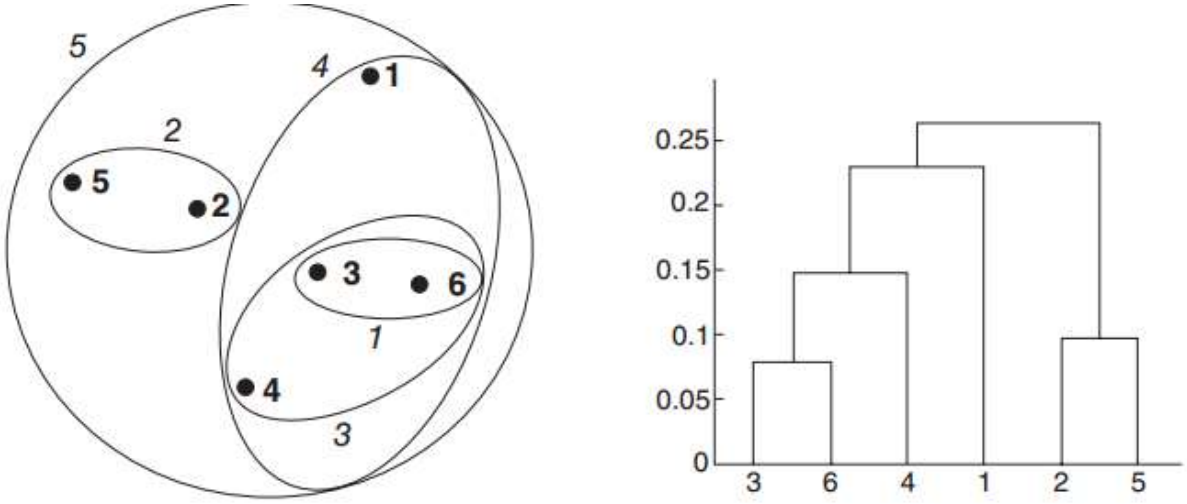
dist({2,5},{1}) = (0.2357 + 0.3421)/(2*1)
= 0.2889.

dist({3,6,4}, {2,5}) = (0.15 + 0.28 + 0.25 + 0.39 + 0.2 + 0.29)/(6*2)
= 0.26.

Через те, що відстань dist({3,6,4}, {2,5}) менша ніж dist({3,6,4}, {1}) і відстань dist({2,5},{1}), кластери {3,6,4} і {2,5} об’єднуються на четвертій стадії.

Для метода Варда близькість між двома кластерами визначається як приріст квадрату помилку після об’єднання двох кластерів. Таким чином, цей метод використовує таку саму цільову функцію як і К-середніх. Це означає, що ця властивість робить метод Варда особливою технікою ієрархічної кластеризації, математично можна показати, що метод Варда дуже схожий на group average.

Рисунок 3.16 показує результат застосування методу Варда до шести точок.



(a) Ward's clustering. (b) Ward's dendrogram.

Рисунок 3.16 – Кластеризація методом Варда

Метод центроїдів розраховує близькість між двома кластерами як відстань між їх центроїдами. Метод центроїдів має характеристику, що не є притаманною ієрархічній кластеризації: можливість інверсій. Два кластери, що були об'єднані можуть бути більш схожими ні ж ті, що були об'єднані на попередньому кроці. Для інших методів відстань між об'єднаними кластерами монотонно зростає в процесі переходу від одиничних кластерів до єдиного кдастера.

Будь-яка з мір близькості кластерів, що були обговорені вище може бути розглянута як вибір різних параметрів (у формулі Ленса-Вільямса) для близькості між кластерами Q та R, де R – сформований об'єднанням кластерів A та B. В цьому виразі p(.,.) – це функція близькості, де m_A, m_B, m_Q - кількість точок в кластерах A, B, Q відповідно. Іншими словами, після об'єднання кластерів A та B формується кластер R і близкість нового кластера R до існуючого кластера Q - це лінійна функція близькостей Q до кластерів A та B.

$$p(R,Q) = \alpha_A\, p(A,Q) + \alpha_B\, p(B,Q) + \beta\, p(A,B) + \gamma\, |p(A,Q) - p(B,Q)|$$

Таблиця 3.5 показує значення цих коефіцієнтів для всіх розглянутих вище технік.

Таблиця 3.5 – Коефіцієнти Ленса-Вільямса для різних типів ієрархічної кластеризації

Clustering Method	α_A	α_B	β	γ
Single Link	$1/2$	$1/2$	0	$-1/2$
Complete Link	$1/2$	$1/2$	0	$1/2$
Group Average	$\frac{m_A}{m_A+m_B}$	$\frac{m_B}{m_A+m_B}$	0	0
Centroid	$\frac{m_A}{m_A+m_B}$	$\frac{m_B}{m_A+m_B}$	$\frac{-m_A m_B}{(m_A+m_B)^2}$	0
Ward's	$\frac{m_A+m_Q}{m_A+m_B+m_Q}$	$\frac{m_B+m_Q}{m_A+m_B+m_Q}$	$\frac{-m_Q}{m_A+m_B+m_Q}$	0

Будь-яка техніка ієрархічної кластеризації може бути виражена з використанням формули Ленса-Вільямс. В той час як загальна формула є привабливою, особливо для імплементації, простіше зрозуміти різні ієрархічні методи розглядаючи визначення близькості кластерів для кожного методу окремо.

Агломеративна ієрархічна кластеризація не може розглядатися як глобальна оптимізація цільової функції. Проте техніка агломеративної ієрархічної кластеризації використовує різні критерії локально, на кожному кроці, які кластери повинні бути об'єднані (чи розділені). Цей підхід породжує алгоритми кластеризації, які уникають складності вирішення задачі оптимізації. Більш того такі підходи не мають проблем з локальними мінімумами чи вибором початкових точок. Часова складність $O(m^2 \log m)$ і потреби пам'яті $O(m^2)$ надмірні в деяких випадках.

Одним з аспектів агломеративної ієрархічної кластеризації є відсутність розгляду умовних розмірів пар кластерів, що об'єднуються. Існує два підходи: зважений, що розглядає всі кластери в рівній мірі і незважений, що бере кількість точок в кожному кластері в облік. Терміни зважений і незважений відносяться до точок даних, а не до кластерів. Іншими словами, кластери різного розміру в рівній мірі дають різні ваги точкам в різних кластерах, в той же час коли розгляд розміру кластера дає точкам в різних кластерах одну і ту ж вагу.

В літературі з кластеризації повне ім'я цього підходу – Unweighted Pair Group Method using Arithmetic averages (UPGMA). В таблиці 3.5, що дає

формулу оновлення близькості кластерів, коефіцієнт UPGMA включає розмір кожного з кластерів, що були об’єднані:

$$\alpha_A = \frac{m_A}{m_A+m_B}\alpha_B = \frac{m_B}{m_A+m_B}, \beta = 0, \gamma = 0.$$

Для зваженої версії WPGMA ці коефіцієнти сталі:

$$\alpha_A = 1/2, \alpha_B = 1/2, \beta = 0, \gamma = 0.$$

Дані алгоритми зазвичай використовуються в таких випадках як створення систематизацій та ієрархій. Існують праці, які показують створення даними методами кластерів кращої якості. В той же час агломеративні ієрархічні алгоритми дорожчі в обчисленнях та потребах пам’яті. Факт однозначності об’єднань може спричинити проблеми для зашумлених, багатовимірних даних, такі як документи.

3.2.3 DBSCAN

Кластеризація, що базується на щільності, визначає регіони з високою щільністю, які відділені від інших регіонами з низькою щільністю. DBSCAN – простий і ефективний алгоритм, що демонструє важливі концепти для інших алгоритмів кластеризації, що базуються на щільності розташування.

Існує не так багато підходів для визначення щільності як для знаходження схожості, існує декілька чітких методів. В підході на базі центру щільність оцінюється для окремої точки в наборі даних через підрахунок точок в певному радіусі (Eps) від даної точки. Він включає і саму точку. Дана техніка зображена на рисунку 3.17. Кількість точок в радіусі Eps від точки A рівна 7, включаючи A.

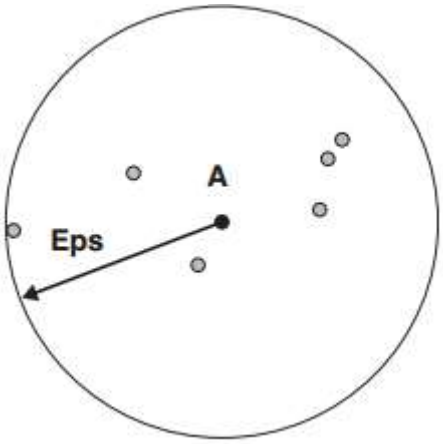


Рисунок 3.17 – Підхід до визначення щільності на базі центру

Цей метод простий в реалізації, але щільність будь-якої точки буде залежати від конкретного радіуса. Якщо радіус достатньо великий, то всі точки матимуть щільність m, кількість точок в наборі даних. Якщо радіус невеликий, то всі точки матимуть щільність 1.

Даний підхід дозволяє класифікувати точки (1) всередині регіону щільності (основні точки), (2) на краю регіону щільності (точки кордону), (3) регіон поділу (фонові точки). Рисунок 3.18 графічно зображає концепцію основних, кордонних та фонових точок, використовуючи колекцію двовимірних точок.

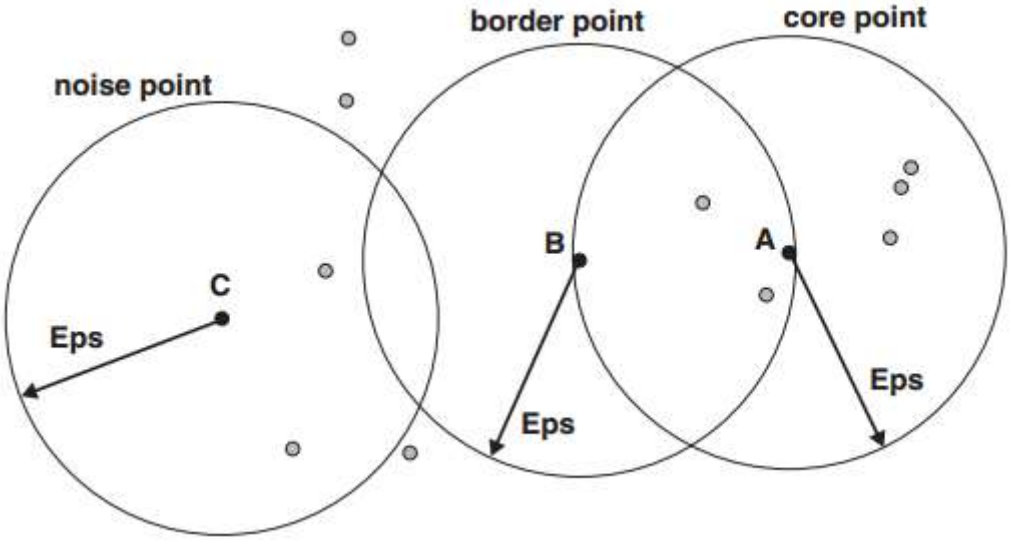


Рисунок 3.18 – Класифікація точок

Основні точки – точки, що знаходяться всередині кластера, який базується на щільності. Точка належить множині основних якщо кількість сусідніх точок, що визначені функцією відстані і визначеним параметром

користувача(Eps), перевищує визначений поріг (MinPts), який також задається користувачем. На рисунку 3.18 точка A є основною для визначеного радіуса (Eps) якщо MinPts ≤ 7.. Точки на кордоні не є основними, але лежать поруч з сусідами основної точки. На рисунку 3.18 точка B знаходиться на кордоні. Фонові точки - ті, які не є ні основними, ні на кордоні. На рисунку 3.18 точка C – фонова.

Після визначення основних, точок на кордоні та фонових точок алгоритм DBSCAN може бути неформально описаний наступним чином. Будь-які дві основні точки, що є достатньо близькими в межах дистанції Eps одна від одної поміщаються в один кластер. Також будь-яка точка на кордоні, що достатньо близька до основної точки, поміщується в цей же кластер основної точки. (Може виникнути проблема якщо точка на кордоні близька достатньо до декількох основних точок з різних кластерів.) Фонові точки відкидаються. Надамо формальне визначення алгоритму, що використовує концепт DBSCAN, але оптимізований для простоти:

1. Позначити точки як основні, на кордоні та фонові.
2. Виключити фонові точки.
3. Прокласти ребро між усіма основними точками радіуса Eps один від одної.
4. Віднести кожну групу об’єднаних основних точок до єдиного кластера.
5. Присвоїти точки на кордоні до одного з кластерів при асоціації з основними точками.

Базова часова складність алгоритму DBSCAN O(m*час для пошуку сусідів в околі Eps), де m - кількість точок. В найгіршому випадку, ця складність рівна $O(m^2)$. В просторі маловимірних даних існують структури даних, такі як kd-дерево, що дозволяють ефективно отримати всі точки на певній відстані від даної і складність може зменшитися до O(mlogm). Потреби пам’яті в DBSCAN навіть при багатовимірних даних рівні O(m), тому що необхідно зберігати тільки невелику кількість даних для кожної точки, тобто

мітку кластера та ідентифікатор кожної точки як основної, на кордоні чи фонової.

Існує проблема вибору параметрів Eps та MinPts. Базовий підхід полягає в огляді поведінки на відстані від точки до її k-го найближчого сусіда, відстань називається k-dist. Для точок, що належать певному кластеру, значення k-dist буде невеликим, якщо k не більше ніж розмір кластера. Існує декілька варіації в залежності від щільності кластера та випадкового розподілу точок, але в середньому кількість варіацій не буде великою, якщо щільності кластерів не відрізняються радикально. Для точок поза кластером, таких як фонові точки, k-dist буде відносно великою. Якщо обчислювати k-dist для всіх точок даних якогось k, відсортувати їх в порядку зростання і потім побудувати графік значеннь, очікуємо побачити різку зміну значення k-dist, що відповідає підходящому значенню Eps. Якщо вибрати цю відстань як Eps параметр і взяти значення k як MinPts, то точки для яких k-dist менша ніж Eps будуть позначені як основні точки, інші ж будуть точками на кордоні чи фоновими. Рисунок 3.19 показує набір даних, для якого граф k-dist даний на риснуку 3.20.

Рисунок 3.19 – Набір даних для кластеризації

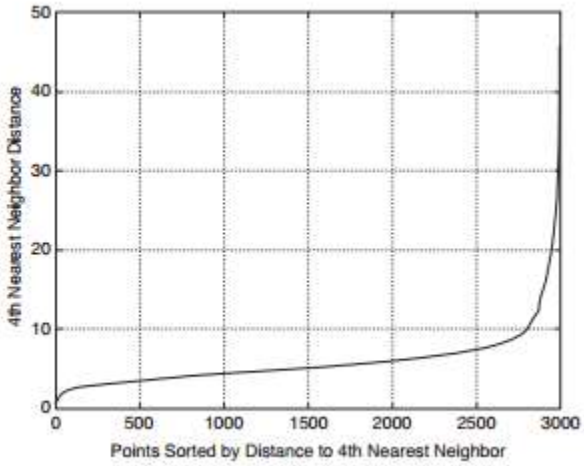


Рисунок 3.20 – Графік k-dist

Значення Eps залежить від k, але не змінюється так радикально як k. Якщо значення k занадто мале, то навіть мала кількість близько розташованих точок, що є шумом чи викидами, будуть невірно позначені як належні кластеру. Якщо значення k занадто велике, то малі кластери (розміру менше k) будуть позначені як шум. Класичний DBSCAN використовує k=4, що є обґрунтованим для двовимірних даних.

DBSCAN має проблеми з щільністю, якщо вона в радикально різна в кластерах. Рисунок 3.21 показує 4 кластери посеред шуму.

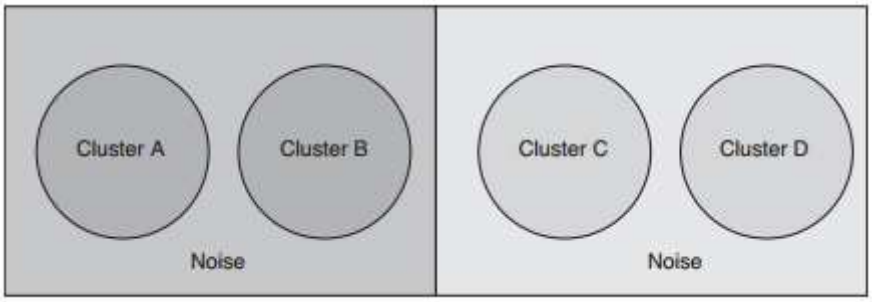
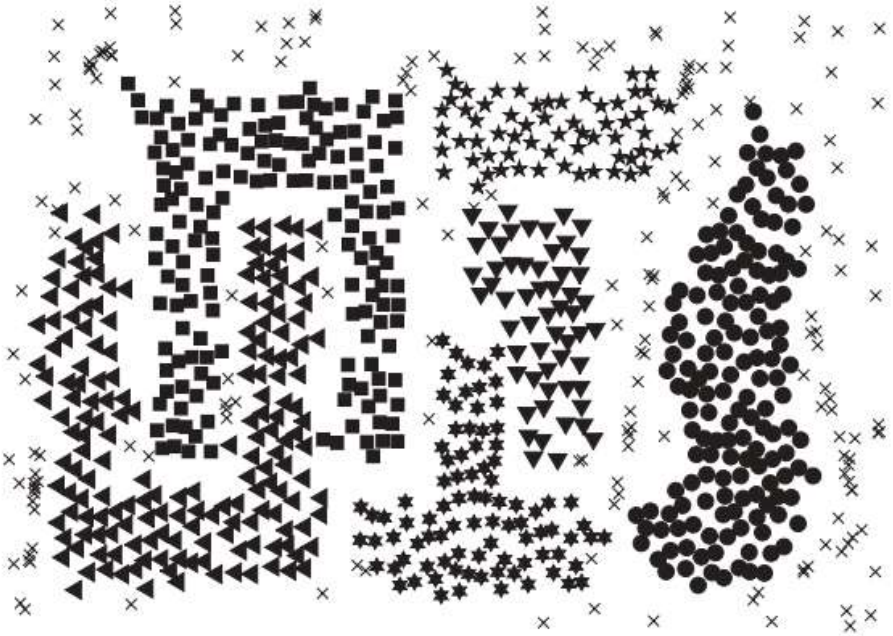


Рисунок 3.21 – Чотири кластери посеред шуму

Щільність кластерів та зашумленість регіонів визначаються їх кольором. Шум навколо пари більш щільних кластерів A та B має таку ж саму щільність як кластери C та D. Якщо Eps достатньо невеликий, то DBSCAN знайде C та D як кластери, потім A та B і точки, що оточують їх, будуть єдиним кластером. Якщо поріг Eps достатньо великий, щоб DBSCAN знайде A та B як окремі

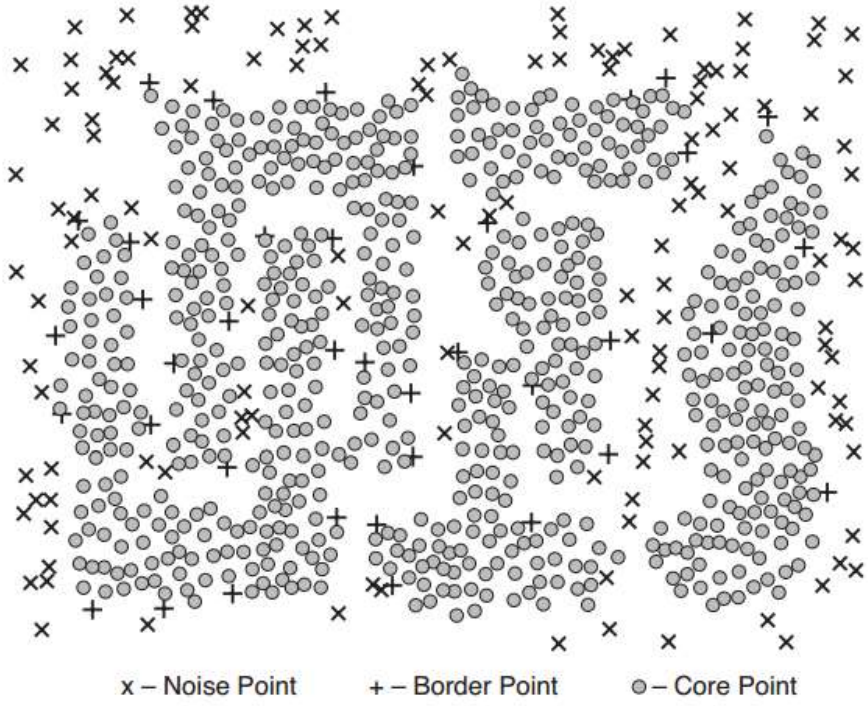
кластери і точки, що їх оточують будуть шумом, то C та D і точки, що їх оточують будуть також шумом.

Для ілюстрації використання DBSCAN, покажемо кластери, що алгоритм знайшов в відносно складних двовимірних даних з рисунку 3.19. Даний рисунок складається з 3000 двовимірних точок. Eps поріг для цих даних був знайдений побудовою відсортованих відстаней чотирьох найближчих сусідів кожної точки і визначенням значення при якому він різко зростає. Було вибрано Eps = 10, що відповідає перегину кривої. Кластери знайдені DBSCAN з параметрами MinPts = 4 та Eps = 10 показані на рисунку 3.22. Основні точки, точки на кордоні та фонові точки показані на рисунку 3.23.



(a) Clusters found by DBSCAN.

Рисунок 3.22 – Кластери знайдені DBSCAN для 3000 точок



(b) Core, border, and noise points.

Рисунок 3.23 – Розподіл точок DBSCAN

Через те, що DBSCAN використовує визначення кластера, що базується на щільності, він відносно стійких до шумів та може опрацьовувати кластери довільної форми та розмірів. DBSCAN може знайти багато кластерів, що не можуть бути знайдені алгоритмом К-середніх. В той же час DBSCAN має проблеми з кластерами, щільність яких відрізняється радикально. Також виникають проблеми з багатовимірними даними, тому що важче визначити щільність таких даних. DBSCAN досить затратний при обчисленні найближчих сусідів.

3.3 Порівняння алгоритмічних реалізацій методів кластерного аналізу

Алгоритми на яких буде проведено порівняння:

- К-середніх;
- Алгоритм ієрархічної кластеризації;
- SOM (Self-Organization Map);
- EM (Expectation Maximization).

Порівняння алгоритмів проведемо на основі наступних факторів:

- Розмір набору даних;
- Кількість кластерів;
- Вид набору даних;
- Тип програмного забезпечення.

Процес порівняння почнемо з визначення факторів, на основі який буде здійснюватися процес (Таблиця 3.6).

Таблиця 3.6 – Фактори порівняння алгоритмів

	Розмір набору даних	Кількість кластерів	Тип набору даних
К-середніх	Великий/малий набір даних	Велика/мала кількість кластерів	Ідеальний/ реальний набір даних
Ієрархічний	Великий/малий набір даних	Велика/мала кількість кластерів	Ідеальний/ реальний набір даних
SOM	Великий/малий набір даних	Велика/мала кількість кластерів	Ідеальний/ реальний набір даних
EM	Великий/малий набір даних	Велика/мала кількість кластерів	Ідеальний/ реальний набір даних

Однією з характеристик, якою можна керувати в процесі здійснення кластерного аналізу, є кількість кластерів, яку бажано отримати в результаті. Різні алгоритми по різному реагують на кількість кластерів. Зміна продуктивності продиктована кількістю кластерів наведена в таблиці 3.7.

Таблиця 3.7 – Залежність продуктивності алгоритмів в залежності від кількості кластерів

	Продуктивність

Кількість кластерів	SOM	К-середніх	EM	Ієрархічний
8	59	63	62	65
16	67	71	69	74
32	78	84	84	87
64	85	89	89	92

Побудуємо графіки залежності продуктивності алгоритмів від кількості кластерів та порівняємо результати (Рисунок 3.24).

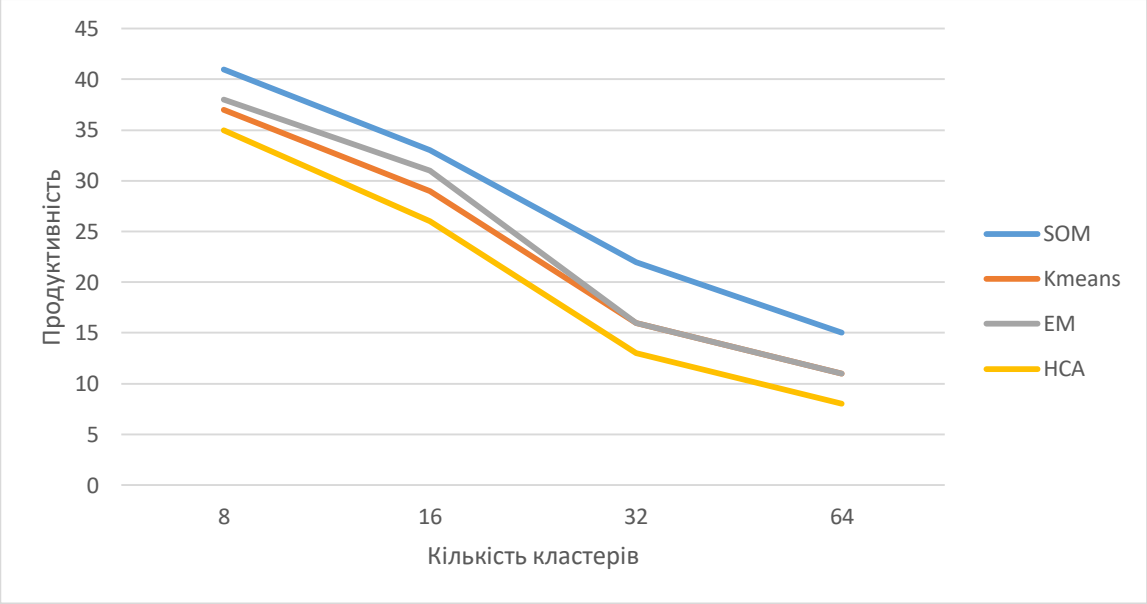


Рисунок 3.24– Графік залежності продуктивності алгоритмів в залежності від кількості кластерів

Зі зростанням кількості кластерів продуктивність SOM алгоритму падає недостовірно. Продуктивність EM та К-середніх стає кращою ніж в ієрахічного алгоритму.

Кількість кластерів впливає також на якість кластеризації, тобто достовірність отриманих результатів (Таблиця 3.8).

Таблиця 3.8 - Залежність якості кластеризації в залежності від кількості кластерів

Кількість	Якість			
кластерів	SOM	К-середніх	EM	Ієрархічний

8	1001	1112	1101	1090
16	920	1089	1076	960
32	830	910	898	850
64	750	840	820	760

Нижче зображено порівняння зміни якості кластеризації при зміні кількості кластерів для вибраних алгоритмів (Рисунок 3.25).

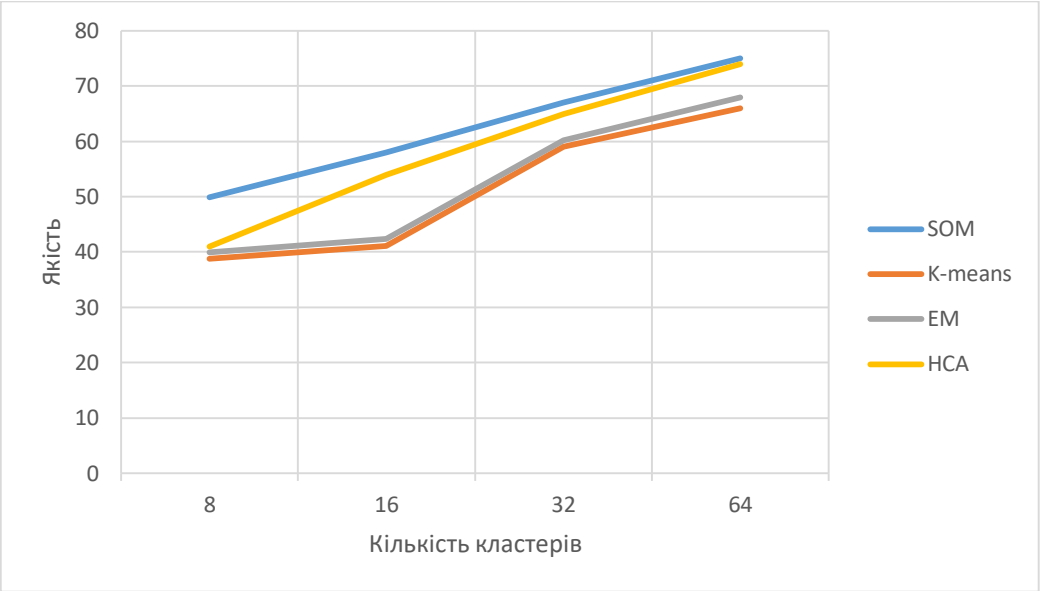


Рисунок 3.25 – Графік залежності якості алгоритмів в залежності від кількості кластерів

Відповідно до графіку найбільшу точність класифікації показує алгоритм SOM. Зі зростанням кількості кластерів точність ієрархічної кластеризації стає більшою і досягає SOM. К-середніх та EM мають меншу точність за інші алгоритми.

Дані, з якими працює алгорим є одним з найбільш впливових факторів на процес кластеризації. Розглянемо вплив розміру даних на діяльність алгоритмів (Таблиця 3.9).

Таблиця 3.9 – Вплив розміру даних на алгоритми

К = 32

Розмір даних	SOM	К-середніх	EM	Ієрархічний
36000	830	910	898	850

4000	89	95	93	91

Результати порівняння наведемо на гістограмах (Рисунок 3.26, 3.27).

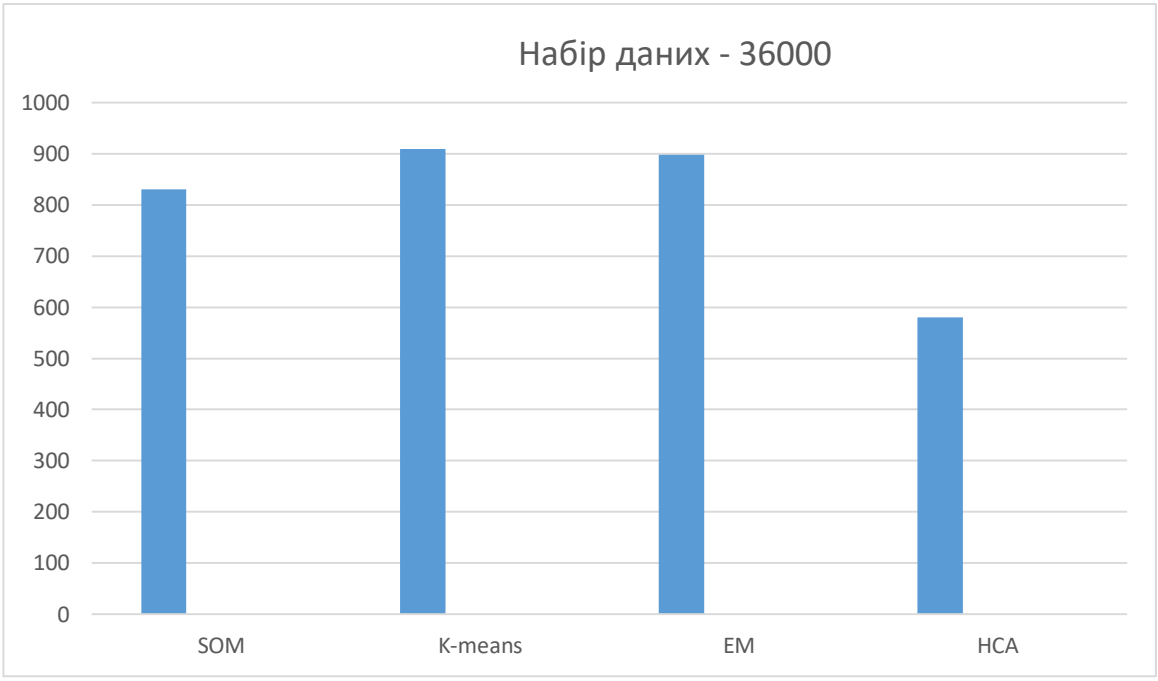


Рисунок 3.26 – Порівняння якості алгоритмів на великому наборі даних

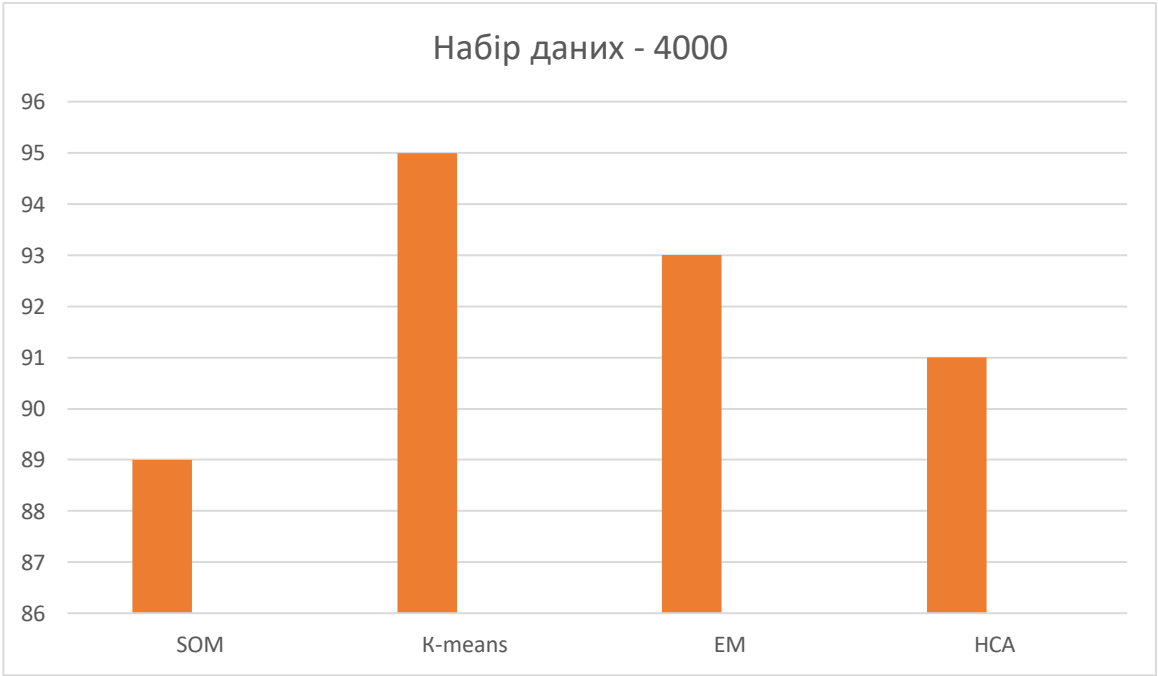


Рисунок 3.27 – Порівняння якості алгоритмів на малому наборі даних

Великий набір даних містить 600 рядків та 60 стовпців, маленький набір – 200 рядків та 20 стовпців. Якість EM та K-середніх стає дуже гарною при використанні великого набору даних. Інші два алгоритми: ієрархічний та SOM показують гарні результати на малих наборах даних.

Не тільки розмір даних є впливовим на алгоритми, а також їх формат (Таблиця 3.10).

Таблиця 3.10 – Вплив типу даних на алгоритми

К = 32

Тип даних	SOM	K-середніх	EM	Ієрархічний
Випадкові	830	910	898	850
Ідеальні	798	810	808	829

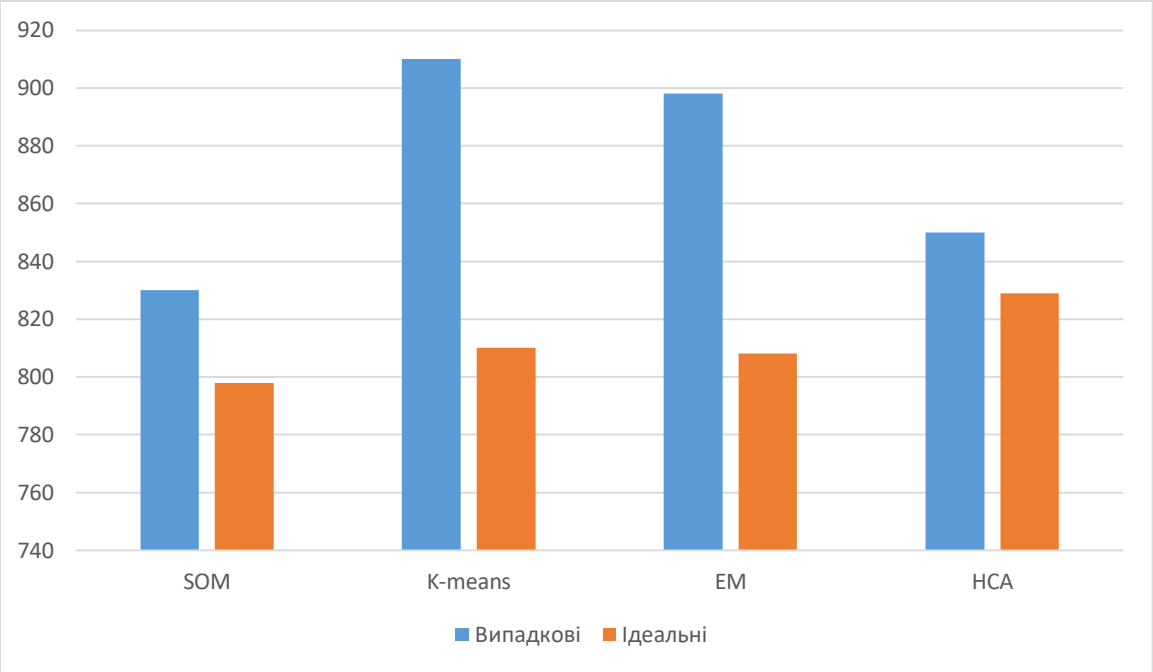


Рисунок 3.28 – Залежність якості роботи алгоритмів від типу даних

Ієрархічна кластеризація та SOM алгоритм дають кращі результати ніж К-середніх та EM при використанні випадкового набору. Це показує те, що К-середніх та EM алгоритми дуже чутливі до шумів в наборі даних. Ієрархічний алгоритм більш чутливий до шумів ніж SOM.

Після аналізу результатів тестування алгоритмів кластеризації можна зробити наступні висновки:

- Зі збільшенням кількості кластерів продуктивність SOM алгоритму зменшується;
- Продуктивність алгоритмі К-середніх та EM кращі ніж ієрархічний алгоритм кластеризації;

- SOM алгоритм показує кращу точність в класифікації ніж інші кластеризації;
- Зі збільшенням кількості кластерів точність ієрархічної кластеризації покращується і досягає алгоритму SOM;
- Алгоритми К-середніх та ЕМ мають гіршу якість (точність) ніж інші.
- Всі малгоритми мають деяку неоднозначність в зашумлених даних;
- Якість ЕМ та К-середніх алгоритмів стає дуже хорошою на великих наборах даних;
- Ієрархічна кластеризація і SOM алгоритми показують гарні результати на невеликих наборах даних;
- Частинні алгоритми (К-середніх, SOM) рекомендовані для великих наборів даних, ієрархічні алгоритми кращі для невеликих наборів;
- Ієрархічні алгоритм та SOM дають кращі результати на випадкових наборах і навпаки;
- К-середніх та ЕМ алгоритми дуже чутливі до шумів в наборах даних;
- Ієрархічна кластеризація більш чутлива до шумів ніж SOM алгоритм.

ВИСНОВКИ ДО РОЗДІЛУ 3

У даному розділі були розглянуті алгоритми кластерного аналізу та проведено їх порівняння. Для подальшої реалізації системи обрано алгоритми К-середніх та ієрархічної класифікації Уорда.

4 ПРОЕКТУВАННЯ СИСТЕМИ КЛАСТЕРНОГО АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ НАУКОВО-ТЕХНІЧНИХ ДАНИХ

4.1 Структура системи

Система кластерного аналізу складається з чотирьох основних блоків: блок аналізу, блок кластеризації, блок оцінки якості та інтерпретації кластерного рішення, засоби візуалізації. Структура системи приведена на рисунку 4.1.

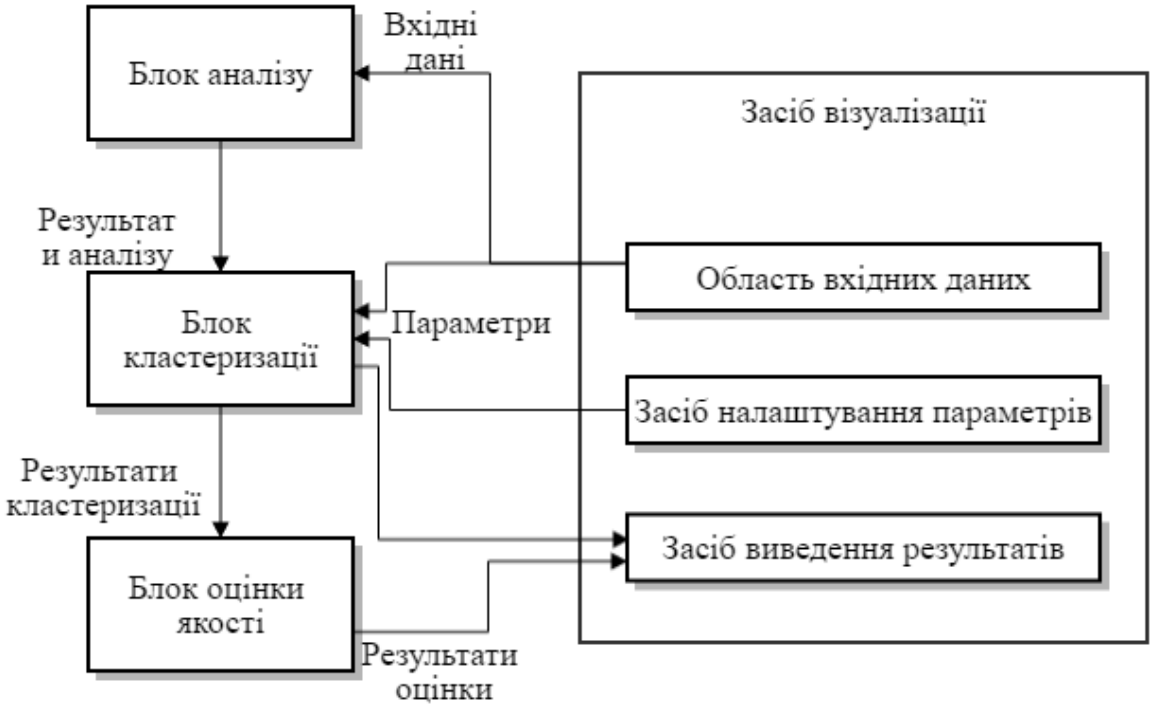


Рисунок 4.1 – Структура системи

Розглянемо призначення кожного блоку. Блок аналізу необхідний для виконання попереднього аналізу вхідних даних. А саме, перед початком кластеризації системі мають бути відомі такі відомості як кількість рядків в файлі, кількість і тип змінних (кількісний, якісний, номінальний) і т.д. Це виконується на етапі попереднього аналізу. Блок кластеризації використовується для безпосереднього розподілу об’єктів до кластерів. В цьому блоці проводиться обчислення степені схожості між об’єктами. Метод

обчислення міри схожості може різним і залежати від налаштувань системи. Після отримання матриці схожості блок кластеризації розподіляє об’єкти в кластери. Вибір алгоритму кластеризації та його параметрів здійснюється в налаштуваннях системи. Планується реалізація різних типів алгоритмів кластерного аналізу даних. Засіб оцінки якості кластеризації призначений для оцінки степеню достовірності кластерних рішень, на основі яких будуть видані відповідні рекомендації. Засіб візуалізації представляє користувачу можливість взаємодії з системою.

Виходячи з вимог до системи кластерного аналізу та візуалізації даних та структурної будови самої системи була побудована концептуальна модель зображена на рисунку 4.2.

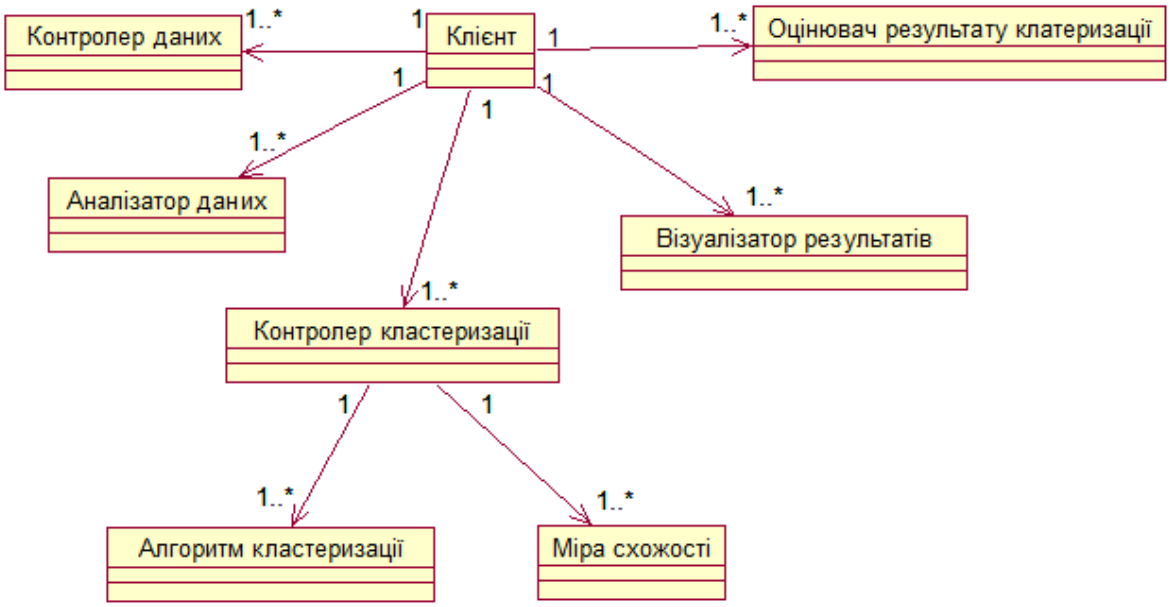


Рисунок 4.2 – Концептуальна модель системи

Модель на рисунку 4.2 є одним із варіантів представлення діаграми об’єктів, що застосовують при моделюванні статичних видів системи з точки зору проектування та процесів. Діаграма об’єктів представляє собою знімок екземплярів класів системи в будь-який період часу. Діаграму об’єктів використовують щоб відобразити певний варіант конфігурації елементів системи. Діаграма об’єктів зручна для відображення зв’язків між об’єктами. В багатьох випадках структуру зв’язків можна визначити за допомогою діаграми

класів, але тоді вона стає складною для розуміння. Не зважаючи на допоміжну функцію, декілька діаграм об’єктів допомагають вирішити цю проблему.

4.2 Діаграма прецедентів

Діаграма прецедентів використовується для об’єднання вимог до системи включаючи внутрішні та зовнішні впливи. Вимоги є здебільшого функціональними. Тобто, коли система аналізується для визначення функціональності, використовуються прецеденти.

Цілями діаграми прецедентів є:

- об’єднання вимог до системи;
- отримання стороннього погляду на систему;
- ідентифікація внутрішніх і зовнішніх впливів;
- демонстрація зв’язків між вимогами та акторами.

На рисунку 4.3 відображено функціональну модель системи кластерного аналізу та візуалізації науково-технічних даних.

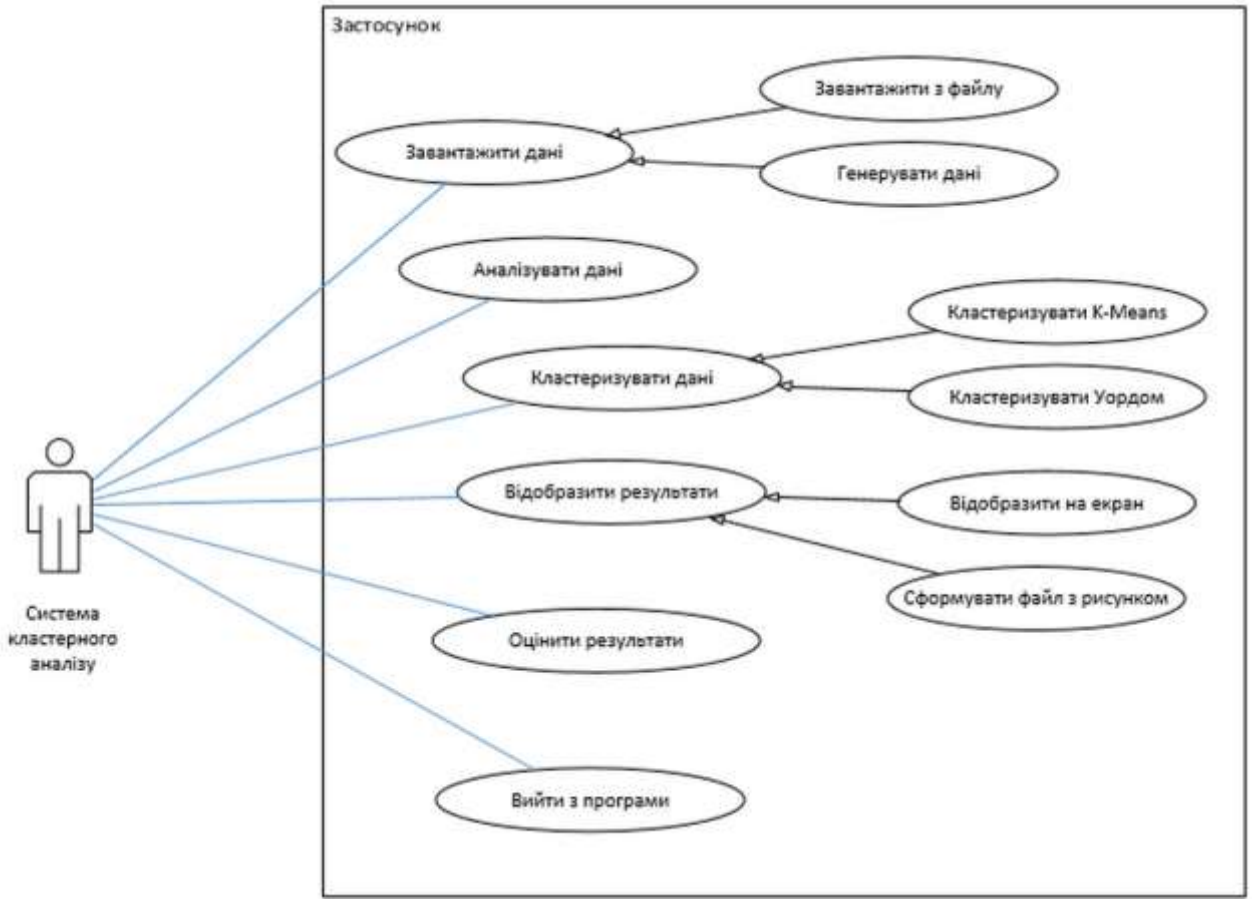


Рисунок 4.3 – Діаграма прецедентів системи

За вибором користувача дані завантажуються в систему чи автономно генеруються системою. Завантажені дані підлягають аналізу для визначення коректності, розмірності, кількості. Аналіз допомагає користувачеві зрозуміти дані та отримати про них додаткову інформацію. Далі виконується процес кластеризації з вико3ристанням вибраних користувачем алгоритмів. Результати кластерного аналізу відображаються користувачеві і можуть бути збережені при потребі.

4.3 Діаграма послідовності

Взаємодія між об’єктами системи полягає в обміні інформацією між ними (в вигляді повідомлень). Особливістю повідомлень є те, що вони не тільки несуть певну інформацію до отримувача, а й можуть певним-чином модифікувати його. Діаграма послідовності відноситься до класу діаграм взаємодії і розглядає аспекти поведінки системи з плином часу.

На основі розробленої вище діаграми прецедентів створена діаграма послідовності (Рисунок 4.4).

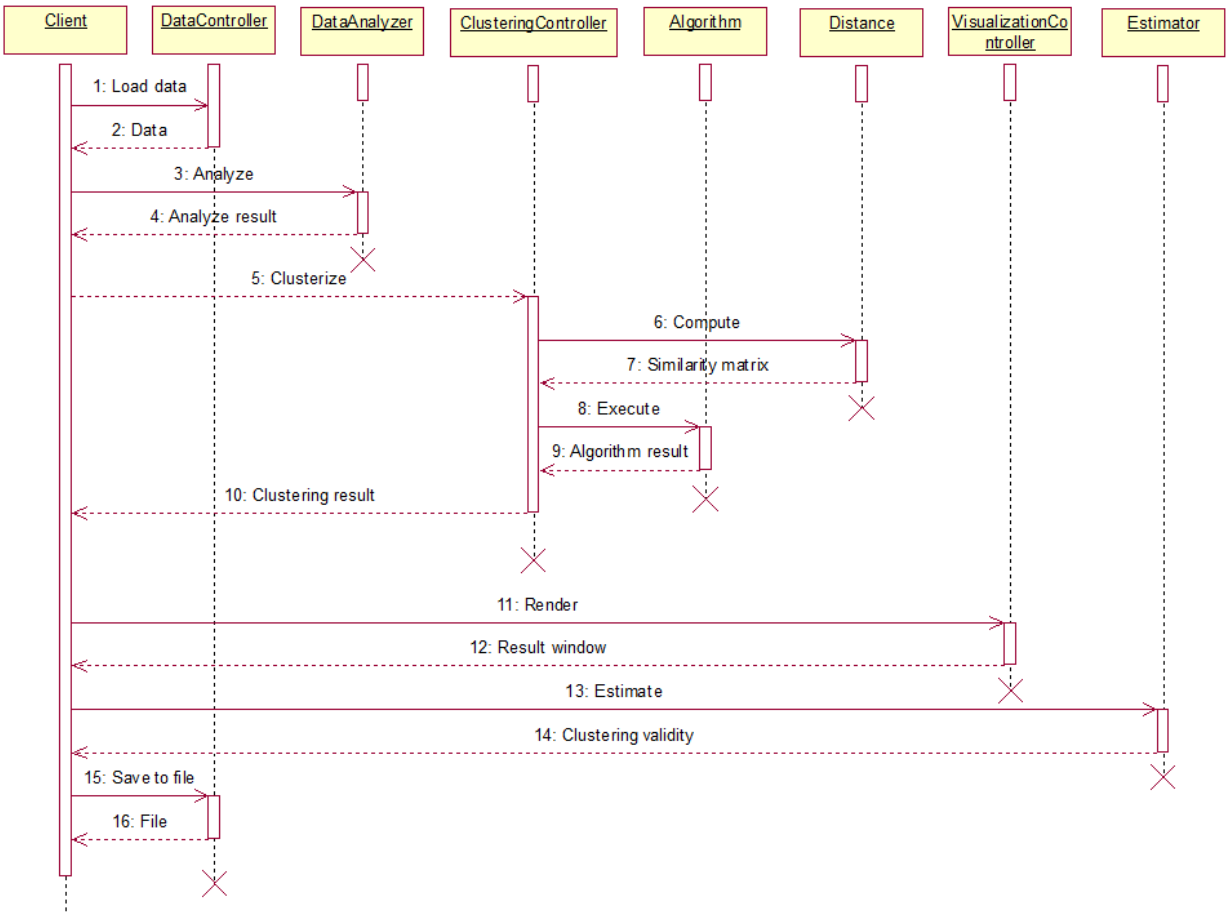


Рисунок 4.4 – Діаграма послідовності системи

На діаграмі послідовності об’єктами виступають здебільшого екземпляри класів або користувачі, що ініціюють певну діяльність. В даному випадку на діаграмі, що відображена на рисунку 4.4, показані часові послідовності виклики певних функцій розроблюваної системи. Таким чином, на діаграмі послідовності можемо побачити такі аспекти:

- повідомлення, що примушують об’єкти виконувати якусь дію;
- дії, які ініціюються повідомленнями (методи) – здебільшого це передача повідомлення іншому об’єкту чи повернення певних даних;
- послідовність обміну повідомленнями в системі.

4.4 Діаграма діяльності

Створення системи – процес спуску від загальної концепції системи через освоєння логічної струкрури до детальних моделей, що описують фізичну реалізацію. Діаграма діяльності належить до логічної моделі.

Дія може містити вхідні чи вихідні дуги діяльності, що демонструють потоки керування чи потоки даних. Потік керування з'єднує дві дії. Потік даних закінчується об'єктом.

Дія виконується тільки при готовності всіх її входів. Після виконання дія передає керування чи дані на виходи. Діаграму діяльності прийнято будувати, щоб дії відображалися зверху вниз чи справа наліво.

Для того, щоб показати на якій стадії знаходиться процес використовується спеціальний маркер. Дане поняття вводиться для зручності опису динамічного процесу.

Перехід маркера здійснюється між вузлами. Маркер керування не містить ніякої додаткової інформації. Маркер даних містить посилання на об'єкт чи структуру даних.

Таким чином, діаграма діяльності (Рисунок 4.5) може використовуватися як для опису бізнес-процесу так і для опису функціональних вимог до системи. Ціль концептуального опису – показати цілісну картину бізнес-процесу предметної області. Отже, діаграма діяльності добре відображає: послідовність дій, події, що ініціюють діяльність чи є кінцевим результатом, умови розширення сценарію.

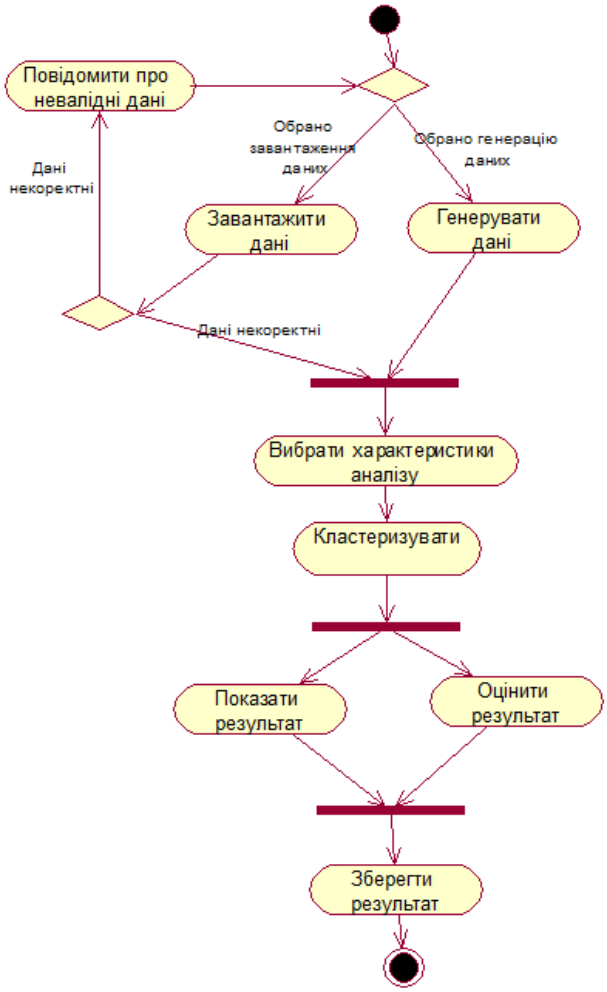


Рисунок 4.5 – Діаграма діяльності системи

4.5 Розробка блок-схем алгоритмів кластеризації

Схема – абстракція якого-небудь процесу чи системи, що більш наглядно відображає значимі частини. Блок-схема являє собою блоків символів, що відповідають етапам роботи алгоритму. Для повного розуміння алгоритмів кластерного аналізу було розроблено набір блок-схем, що відображають всі значимі кроки алгоритмів кластеризації K-means, Уорда. На рисунку 4.6 зображена блок-схему алгомеративної ієрархічної кластеризації методом Уорда.

Рисунок 4.6 – Блок схема алгоритму Уорда

На рисунку 4.7 показна блок-схема іншого більш складного алгоритму кластеризації К-середніх. К-середніх – алгоритм ітеративної кластеризації, тому виконує декілька визначених кроків певну кількість ітерацій.

Рисунок 4.7 – Блок схема алгоритму К-середніх

На рисунку 4.8 зображено блок-схему алгоритму DBSCAN, що відноситься до алгоритмів, що базуються на щільності. Цей алгоритм є більш складним за попередні і складається з більшої кількості кроків.

Рисунок 4.8 – Блок-схема алгоритму DBSCAN

ВИСНОВКИ ДО РОЗДІЛУ 4

У даному розділі були розглянуті основні етапи проектування системи кластерного аналізу науково-технічних даних.

5 РЕАЛІЗАЦІЯ СИСТЕМИ КЛАСТЕРНОГО АНАЛІЗУ ТА ВІЗУАЛІЗАЦІЇ НАУКОВО-ТЕХНІЧНИХ ДАНИХ

5.1 Вибір основних інструментів реалізації

5.1.1 Вибір мови програмування

Для реалізації сервісу була обрана високорівнева мова програмування Python, реалізація інтерпретатора 2.7. Одним з її переваг є легкість для читання і ясність. Програмний код на мові Python читається легше, а значить багаторазове його використання і обслуговування виконується набагато простіше, ніж використання програмного коду на інших мовах.

Крім того, Python підтримує сучасні механізми багаторазового використання програмного коду з арсеналу об'єктно-орієнтованого і функціонального програмування. У порівнянні з компілює або строго типізований мовами, такими як C, C ++ і Java, Python у багато разів підвищує продуктивність праці розробника. Велика частина програм на цій мові виконується без змін на всіх основних платформах. У складі Python поставляється велика кількість зібраних і переносяться функціональних можливостей, що становлять стандартну бібліотеку.

Як середовище розробки сервісу була обрана IDE Sublime Text - досить розвинена і широко відома інтегроване середовище розробки модульних крос-платформних додатків, що є вільним програмним забезпеченням. На користь цього вибору зіграв і той факт, що у автора роботи вже був досвід роботи з Sublime Text. Будучи крос-платформної, система підтримує багато мов програмування, а також дозволяє додавати підтримку нових за допомогою гнучкої системи плагінів. Використовуючи цей функціонал, автором розробки в Sublime Text була додані підтримка мови Python.

Sublime Text являє собою першу настільки потужно підтриману світовим IT-спільнотою спробу створення єдиної відкритої інтегрованої

платформи розробки додатків, що володіє надійністю, функціональністю і рівнем якості комерційного продукту. Фактично, ця платформа призначена для всього і ні для чого конкретно, представляючи собою основу, що має блочну структуру і інтегруючу інструменти розробки ПО різних виробників для створення додатків на будь-якій мові, з використанням будь-яких технологій і для будь-якої програмної платформи. Навколо проекту Sublime Text в даний час сформовано співтовариство найбільших IT-компаній, серед яких Borland, IBM, SAP AG, RedHat і інші.

5.1.2 NLTK (Natural Language Toolkit)

Бібліотека NLTK - пакет бібліотек і програм для символьної і статистичної обробки природної мови, написаних на мові програмування Python. Містить графічні уявлення і приклади даних. Супроводжується великою документацією, включаючи книгу з поясненням основних концепцій, що стоять за тими завданнями обробки природної мови, які можна виконувати за допомогою даного пакету.

NLTK є провідною платформою для створення програм на мові Python для роботи з даними людською мовою. Він забезпечує легкий у використанні інтерфейс для більш 50 корпусів і лексичних ресурсів, таких як WordNet, поряд з набором бібліотек обробки текстів для класифікації, токенізації, що випливають, мічення, синтаксичного і семантичного міркування, обгорток для бібліотек NLP промислової міцності, і активний форум.

5.2 Особливості реалізації програмного забезпечення

В даному розділі наведено декілька прикладів кластеризації наборів документів за допомогою Python. Перший приклад стосується виявлення прихованих структур у описах топ-100 фільмів усіх часів (за списком IMDB).

5.2.1 Підготовка до роботи

По-перше, потрібно імпортувати всі необхідні для роботи бібліотеки:

```
In [4]: import numpy as np
        import pandas as pd
        import nltk
        import re
        import os
        import codecs
        from sklearn import feature_extraction
        import mpld3
```

Використовуються два наступні основні списки:

- *'titles'*: назви фільмів у заданому порядку;
- *'synopses'*: описи фільмів, що відповідають порядку назв.

Першочергове значення має список *'synopses'* ; *'titles'* в основному використовуються для маркування.

```
In [3]: #імпорт 3-х списків: назви, посилання і описи
        titles = open('title_list.txt').read().split('\n')
        #зчитуємо перші 100
        titles = titles[:100]

        links = open('link_list_imdb.txt').read().split('\n')
        links = links[:100]

        synopses_wiki = open('synopses_list_wiki.txt').read().split('\n BREAKS HERE')
        synopses_wiki = synopses_wiki[:100]

        synopses_clean_wiki = []
        for text in synopses_wiki:
            text = BeautifulSoup(text, 'html.parser').getText()
            #видаляємо html форматування та перетворимо в unicode
            synopses_clean_wiki.append(text)

        synopses_wiki = synopses_clean_wiki


        genres = open('genres_list.txt').read().split('\n')
        genres = genres[:100]

        print(str(len(titles)) + ' назв')
        print(str(len(links)) + ' посилань')
        print(str(len(synopses_wiki)) + ' описів')
        print(str(len(genres)) + ' жанрів')
```

```
100 назв
100 посилань
100 описів
100 жанрів
```

5.2.2 Стоп-слова, стеммінг та токенізація

Цей розділ зосереджений на визначенні деяких функцій для маніпулювання описами. По-перше, потрібно завантажити список NLTK англійських стоп-слів. Стоп-слова – це слова типу "a", "the" або "in", які не несуть для нас корисного значення.

```
In [7]: # завантажуємо nltk стоп-слова у змінну 'stopwords'
        #nltk.download()
        stopwords = nltk.corpus.stopwords.words('english')
```

Далі імпортуємо бібліотеку для роботи з SnowBallS Stemmer, який фактично є частиною NLTK.

```
In [8]: # завантаеумо nltk SnowballStemmer у змінну 'stemmer'
        from nltk.stem.snowball import SnowballStemmer
        stemmer = SnowballStemmer("english")
```

Для виконання необхідних операцій визначимо наступні дві функції:

- *tokenize_and_stem*: розбиває опис на список відповідних слів (або лексем), а також токенізує кожну лексему;
- *tokenize_only*: токенізує лише опис.

Обидві ці функції використовуються для створення словника, який стає важливим у випадку, якщо потрібно використовувати стеми для алгоритму, але пізніше їх знову потрібно конвертувати у повні слова для цілей їх подальшого представлення.

```
In [9]: # визначаемо дві функції

def tokenize_and_stem(text):
    # токенізуемо за реченнями, далі за словами
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    # відфільтровуемо всі токени, що не містять літер (цифрові токени, пунктуація)
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    stems = [stemmer.stem(t) for t in filtered_tokens]
    return stems


def tokenize_only(text):
    # токенізуемо за реченнями, далі за словами
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    # відфільтровуемо всі токени, що не містять літер (цифрові токени, пунктуація)
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    return filtered_tokens
```

Далі представлений код дозволяє два словники: з стемами та лише токенізований.

```
In [10]: totalvocab_stemmed = []
         totalvocab_tokenized = []
         for i in synopses:
             allwords_stemmed = tokenize_and_stem(i)
             totalvocab_stemmed.extend(allwords_stemmed)

             allwords_tokenized = tokenize_only(i)
             totalvocab_tokenized.extend(allwords_tokenized)
```

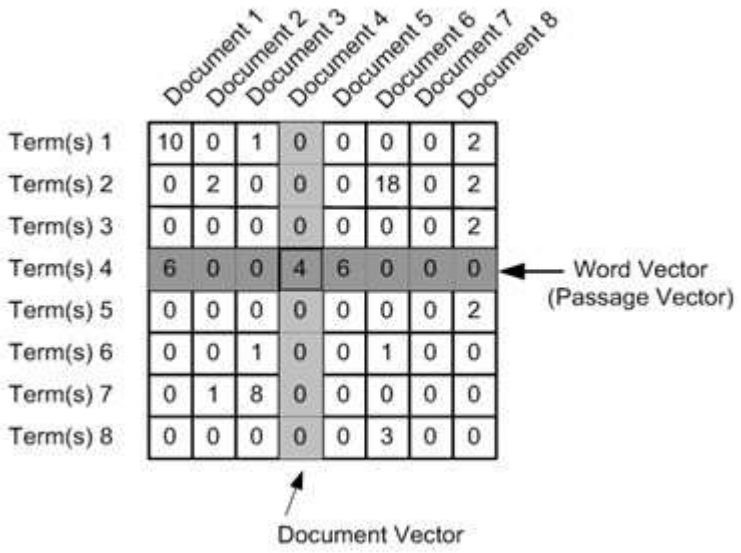
Використовуючи ці два списки та бібліотеку Panda, створюється DataFrame із стемом у якості індексу та токенізованими слова у вигляді стовпця. Перевага використання такої структури полягає в тому, що вона забезпечує ефективний спосіб шукати стем і повертати повне слово. Мінус тут полягає в тому, що стеми до повних слів знаходять у відношенні один до багатьох: стем "біг" може бути пов'язаний з "побіг", "пробіг", "біг" і т.д. Для цілей цієї роботи це нормально.

```
In [11]: vocab_frame = pd.DataFrame({'words': totalvocab_tokenized}, index = totalvocab_stemm
```

5.2.3 Tf-idf і схожість документа

В даному розділі наведено опис створення векторизатора (tf-idf).

З метою отримання матриці Tf-idf потрібно спочатку порахувати входження слів за кожним документом. Це дозволяє отримати матрицю документ-терм (dtm) або матрицю частот термінів:



Потім застосовуємо зважування частоти терму до зворотньої частоти документа: слова, які часто зустрічаються в документі, але не часто в корпусі, отримують більш високу вагу, оскільки вважається, що ці слова мають більше значення для документа.

Деякі параметри, які використовуються наведені нижче:

- max_df: це максимальна частота в документах, яка може використовуватись у матриці tf-idf . Якщо частота появи терму перевищує 80% документів, то він ймовірно є незначним;
- min_idf: число, яке містить кількість документів, в яку повинен входити відповідний терм;

ngram_range: парламент вказує на використання уніграм, біграми та триграм.

```
In [12]: from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=200000,
                                   min_df=0.2, stop_words='english',
                                   use_idf=True, tokenizer=tokenize_and_stem, ngram_range=(1,

%time tfidf_matrix = tfidf_vectorizer.fit_transform(synopses)

print(tfidf_matrix.shape)
```

```
Wall time: 10.4 s
(100, 563)
```

Терми – це лише перелік функцій, які використовуються в матриці tf-idf. Нижче наведений код дозволяє цей перелік подивитись:

```
In [13]: terms = tfidf_vectorizer.get_feature_names()
```

dist визначається як 1 мінус косинусна подібність кожного документа. Косинусна подібність вимірюється відповідно до матриці tf-idf і може бути використана для отримання міри подібності між кожним документом та іншими документами в наборі (кожен опис з кожним). Віднімання його від 1 забезпечує косинусну відстань, яку будемо використовувати для побудови графіків на евклідовій (двовимірній) площині.

Зауважте, що з *dist* можна оцінити схожість будь-яких двох або більше описів.

```
In [14]: from sklearn.metrics.pairwise import cosine_similarity
         dist = 1 - cosine_similarity(tfidf_matrix)
```

5.2.4 K-mean кластеризація

Використовуючи матрицю tf-idf, можна використати ряд алгоритмів кластеризації з метою кращого розуміння прихованої структури в описах. Для початку використаємо алгоритм K-mean. Для даного алгоритму заздалегідь визначається кількість кластерів (в даному прикладі обрано 5). Кожне спостереження присвоюється якомусь кластеру таким чином, щоб мінімізувати внутрішню кластерну суму квадратів. Далі розраховується середнє значення кластерних спостережень і використовується як новий центроїд кластера. Потім спостереження переназначаються кластерам та центроїдам, які перераховуються в ітераційному процесі, поки алгоритм не досягне конвергенції.

In [15]:
```
from sklearn.cluster import KMeans

num_clusters = 5

km = KMeans(n_clusters=num_clusters)

%time km.fit(tfidf_matrix)

clusters = km.labels_.tolist()
```

```
Wall time: 504 ms
```

В роботі використовується бібліотека joblib.dump, яка дозволяє підбирати модель, як тільки вона досягає конвергенції, а також перезавантажувати її і перепризначити мітки новим кластерам.

In [16]:
```
from sklearn.externals import joblib

#joblib.dump(km,  'doc_cluster.pkl')
km = joblib.load('doc_cluster.pkl')
clusters = km.labels_.tolist()
```

Створимо словник назв, рангів, описів, кластерного призначення та жанрів.

Далі, цей словник перетворюємо у Pandas DataFrame для легкого доступу.

In [17]:
```
import pandas as pd

films = { 'назва': titles, 'ранг': ranks, 'опис': synopses, 'кластер': clusters, 'жанр': genres }

frame = pd.DataFrame(films, index = [clusters] , columns = ['ранг', 'назва', 'кластер', 'жанр'])
```

In [18]:
```
frame['кластер'].value_counts()
```

Out[18]:
```
4    26
0    25
2    21
1    16
3    12
Name: кластер, dtype: int64
```

In [19]:
```
grouped = frame['ранг'].groupby(frame['кластер'])

grouped.mean()
```

Out[19]:
```
кластер
0    47.200000
1    58.875000
2    49.380952
3    54.500000
4    43.730769
Name: ранг, dtype: float64
```

Зауважимо, що **кластери 4 і 0** мають найнижчий ранг, що вказує на те, що вони, в середньому, містять фільми, які потрапили до "кращих" у топ-100 списку.

Проіндексуємо та відсортуємо їх за кожним кластером, щоб визначити, які n є часто використовуваними ($n = 6$) словами, найближчими до центру кластера.

```
In [20]: from __future__ import print_function

         print("Верхні терми для кластерів:")
         print()
         order_centroids = km.cluster_centers_.argsort()[:, ::-1]
         for i in range(num_clusters):
             print("Кластер %d слова:" % i, end='')
             for ind in order_centroids[i, :6]:
                 print(' %s' % vocab_frame.ix[terms[ind].split(' ')].values.tolist()[0][0].encode('utf-8', 'ignore'), end=',
             print()
             print()
             print("Кластер %d назви:" % i, end='')
             for title in frame.ix[i]['назва'].values.tolist():
                 print(' %s,' % title, end='')
             print()
             print()
```

```
Верхні терми для кластерів:

Кластер 0 слова: b'family', b'home', b'mother', b'war', b'house', b'dies',
```

5.2.5 Багатовимірне масштабування

Далі наведено код для перетворення матриці dist у двовимірний масив, використовуючи багатовимірне масштабування.

```
In [23]: import os  # for os.path.basename

         import matplotlib.pyplot as plt
         import matplotlib as mpl

         from sklearn.manifold import MDS

         MDS()

         mds = MDS(n_components=2, dissimilarity="precomputed", random_state=

         pos = mds.fit_transform(dist)  # shape (n_components, n_samples)

         xs, ys = pos[:, 0], pos[:, 1]
```

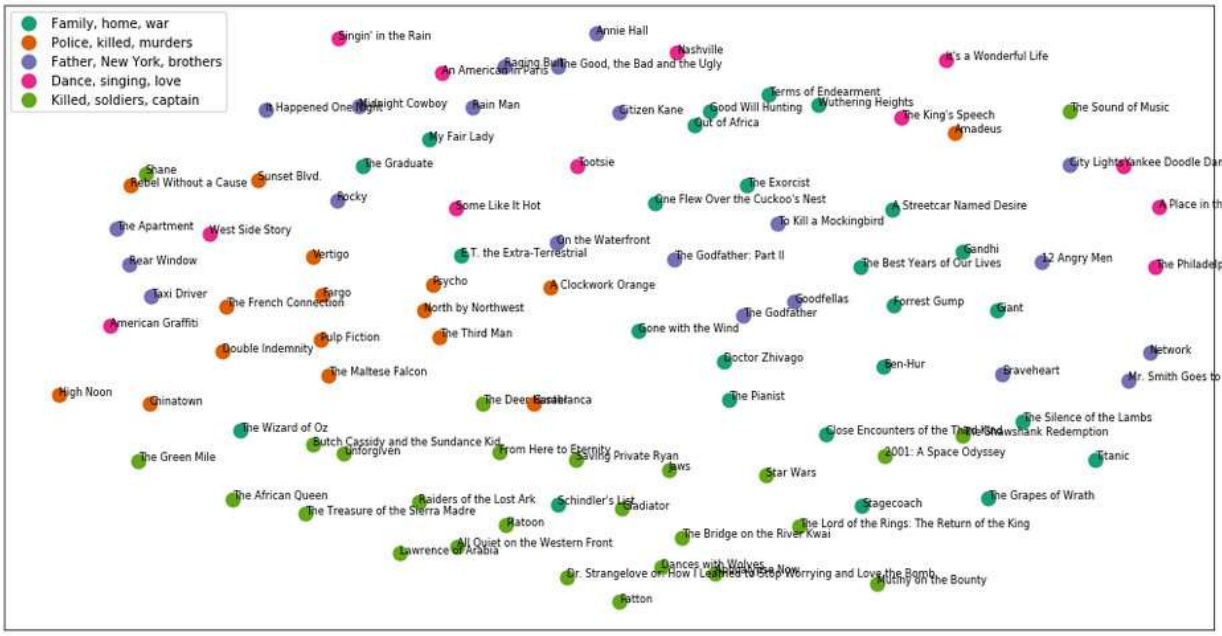
5.2.6 Візуалізація кластерів документів

У цьому розділі продемонстровано, як візуалізувати кластеризацію документів за допомогою matplotlib та mpld3 (обгортка matplotlib для D3.js).

Спочатку визначаємо деякі словники для переходу від номера кластера до кольору та до імені кластера. Назви кластерів обрано на базі слів, найближчих до кожного центру кластера.

In [25]:
```
#задаємо кольори для кластерів через словник
cluster_colors = {0: '#1b9e77', 1: '#d95f02', 2: '#7570b3', 3: '#e7298a', 4: '#66a6

#задаємо назви кластерів через словник
cluster_names = {0: 'Family, home, war',
                 1: 'Police, killed, murders',
                 2: 'Father, New York, brothers',
                 3: 'Dance, singing, love',
                 4: 'Killed, soldiers, captain'}
```

In [27]:
```
#створюємо фрейм даних
df = pd.DataFrame(dict(x=xs, y=ys, label=clusters, title=titles))

#групуємо за кластерами
groups = df.groupby('label')


# створюємо область рисування
fig, ax = plt.subplots(figsize=(17, 9)) # задаємо її розмір
ax.margins(0.05)

#ітеруємо групами для створення рисунку
for name, group in groups:
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=12, label=cluster_names[name],
    ax.set_aspect('auto')
    ax.tick_params(\
        axis= 'x',
        which='both',
        bottom='off',
        top='off',
        labelbottom='off')
    ax.tick_params(\
        axis= 'y',
        which='both',
        left='off',
        top='off',
        labelleft='off')

ax.legend(numpoints=1)  #відтворюємо легенду

#додаємо мітки в x,y позиції з міткою назвою фільму
for i in range(len(df)):
    ax.text(df.ix[i]['x'], df.ix[i]['y'], df.ix[i]['title'], size=8)



plt.show() #відображаємо рисунок

#відкоментувати для можливості збереження рисунку у файл
#plt.savefig('clusters_small_noaxes.png', dpi=200)
```




```
In [52]: plt.close()
```

Кластеризація виглядає чудово, але можна побачити накладки. Для їх прибирання використаємо обгортку для matplotlib. Вона має гарний функціонал для збільшення та панорамування. Нижче наведено Javascript сніппет, який визначає для користувача зону відображення з можливостями зуму/панорамування.

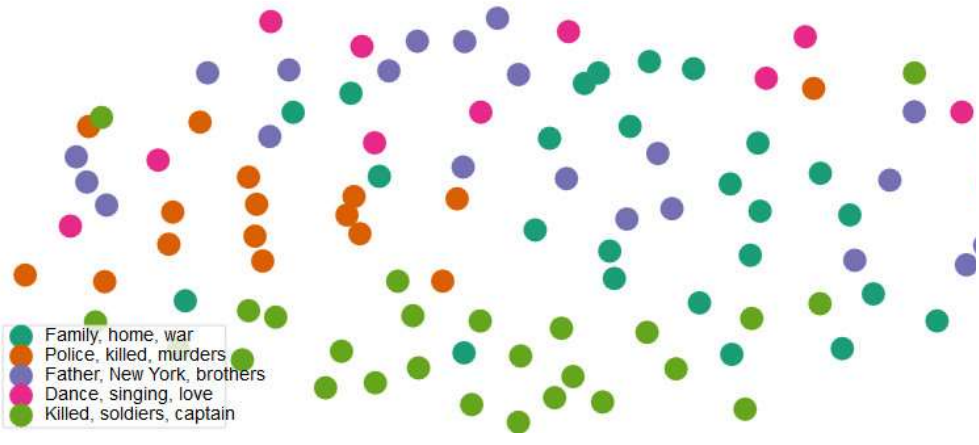
In [29]:
```
#визначаємо клієнтське місцезнаходження тулбару
class TopToolbar(mpld3.plugins.PluginBase):
    """Plugin for moving toolbar to top of figure"""

    JAVASCRIPT = """
    mpld3.register_plugin("toptoolbar", TopToolbar);
    TopToolbar.prototype = Object.create(mpld3.Plugin.prototype);
    TopToolbar.prototype.constructor = TopToolbar;
    function TopToolbar(fig, props){
        mpld3.Plugin.call(this, fig, props);
    };

    TopToolbar.prototype.draw = function(){
      // the toolbar svg doesn't exist
      // yet, so first draw it
      this.fig.toolbar.draw();

      // then change the y position to be
      // at the top of the figure
      this.fig.toolbar.toolbar.attr("x", 150);
      this.fig.toolbar.toolbar.attr("y", 400);

      // then remove the draw function,
      // so that it is not called again
      this.fig.toolbar.draw = function() {}
    }
    """
    def __init__(self):
        self.dict_ = {"type": "toptoolbar"}
```

Зауважте, що mpld3 дозволяє визначити деякий користувальницький CSS, який можна використовувати для стилювання шрифту, осей та лівого поля на рисунку.

In [30]:

```
#створюємо фрейм даних
df = pd.DataFrame(dict(x=xs, y=ys, label=clusters, title=titles))

#групуємо за кластерами
groups = df.groupby('label')

#визначаємо користувацький css для форматування шрифту та видалення назв осей
css = """
text.mpld3-text, div.mpld3-tooltip {
  font-family:Arial, Helvetica, sans-serif;
}

g.mpld3-xaxis, g.mpld3-yaxis {
display: none; }
"""

# Рисунок
fig, ax = plt.subplots(figsize=(14,6)) #задаємо розмір рисунку
ax.margins(0.03)

for name, group in groups:
    points = ax.plot(group.x, group.y, marker='o', linestyle='', ms=18, label=cl
    ax.set_aspect('auto')
    labels = [i for i in group.title]

    #задаємо підказки використовуючи точки, назви та вже визначений 'css'
    tooltip = mpld3.plugins.PointHTMLTooltip(points[0], labels,
                                       voffset=10, hoffset=10, css=css)
    #звязуємо підказки з рисунком
    mpld3.plugins.connect(fig, tooltip, TopToolbar())

    ax.axes.get_xaxis().set_ticks([])
    ax.axes.get_yaxis().set_ticks([])

    #виключаємо назви осей
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)


ax.legend(numpoints=1) #відображаємо легенду

mpld3.display() #відображаємо створений рисунок

#відкоментувати для експорту в html
#html = mpld3.fig_to_html(fig)
#print(html)
```

Out[30]:

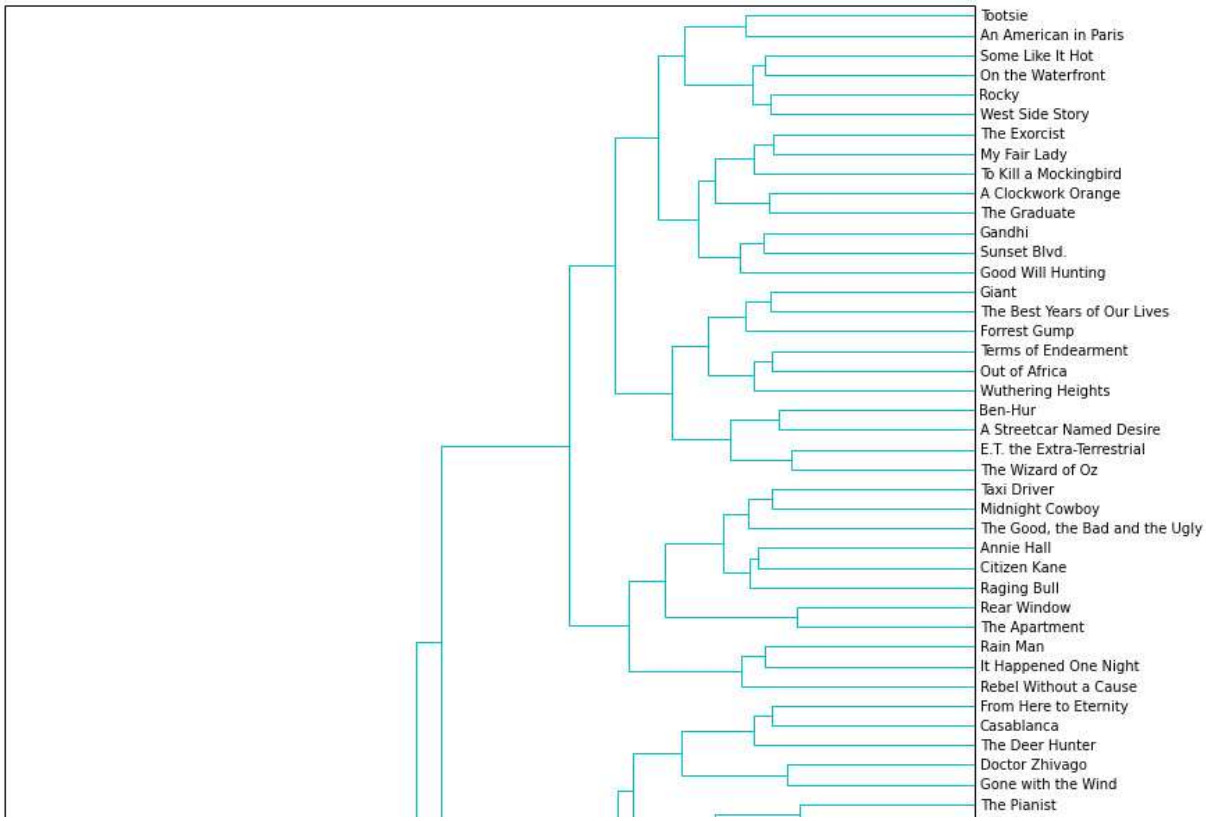


5.2.7 Ієрархічна кластеризація документів

У цьому розділі продемонстровано використання ієрархічного алгоритму кластеризації Уорда – агломераційного методу кластеризації, який визначає, що на кожному етапі пари кластерів з мінімальними відстанями між кластерами об'єднуються. В даному прикладі використовується матриця косинусних відстаней.

Зауважте, що цей метод повернув 3 первинні кластери, причому найбільший кластер розділився на близько 4 основних підкластери. Зауважимо, що кластер червоного кольору містить багато фільмів групи "Killed, soldiers, love". Наприклад, *Хоробре Серце* та *Гладіатор* знаходяться в одному з кластерів нижчого рівня.

In [31]:
```
from scipy.cluster.hierarchy import ward, dendrogram

linkage_matrix = ward(dist) #визначаємо матрицю дистанцій за допомогою алгоритму Уорд

fig, ax = plt.subplots(figsize=(15, 20))
ax = dendrogram(linkage_matrix, orientation="right", labels=titles);

plt.tick_params(\
    axis= 'x',
    which='both',
    bottom='off',
    top='off',
    labelbottom='off')

plt.tight_layout() #будуємо дендрограму
#відкоментувати для можливості збереження діаграми у файл
```



5.2.8 Прихований розподіл Діріхле

Цей розділ присвячений використанню прихованого розподілу Діріхле (LDA), який дозволяє дізнатися більше про приховану структуру. LDA - це імовірнісна тематична модель, яка передбачає, що документи є сумішшю тем і що кожне слово в документі можна віднести до тем документа.

Для реалізації LDA використано пакет GenSim. Нижче наведено функцію, яка видаляє в описах будь-які власні іменники.

```
In [32]: #видаляємо власні іменники
         import string
         def strip_proppers(text):
             # токенізуємо за реченнями, потім за словами
             tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent) if word.is
             return "".join([" "+i if not i.startswith("'") and i not in string.punctuation else i for i in toke
```

Оскільки вищевказана функція ґрунтується на використанні великих літер, вона схильна видаляти слова на початку речень. Інший варіант функції без цієї вади наведена нижче і використовує модуль визначення частин мови NLTK.

```
In [33]: #видалаямо власні іменники
from nltk.tag import pos_tag

def strip_proppers_POS(text):
    tagged = pos_tag(text.split()) #використовуємо модуль визначення частин мови NLT
    non_propernouns = [word for word,pos in tagged if pos != 'NNP' and pos != 'NNPS'
    return non_propernouns
```

Виконуємо обробку тексту (видалення власних іменників, токенізація, видалення стоп-слів):

```
In [35]: #Реалізація прихованого розподілу Діріхле засобами GenSim

from gensim import corpora, models, similarities

#видаляемо власні іменники
preprocess = [strip_proppers(doc) for doc in synopses]

%time tokenized_text = [tokenize_and_stem(text) for text in preprocess]

%time texts = [[word for word in text if word not in stopwords] for text in tokenized_tex
Wall time: 5.36 s
Wall time: 544 ms
```

```
In [37]: dictionary = corpora.Dictionary(texts)

In [38]: dictionary.filter_extremes(no_below=1, no_above=0.8)

In [39]: corpus = [dictionary.doc2bow(text) for text in texts]
```

Створення моделі наведено нижче. Для моделі обрано 100 проходів для забезпечення конвергенції. Як можна побачити, для роботи машини знадобилося 1 хвилина 18 секунд.

```
In [41]: %time lda = models.LdaModel(corpus, num_topics=5, id2word=dict
         Wall time: 1min 18s
```

Кожна тема має набір слів, що визначає її разом з певною вірогідністю.

Перетворимо список найкращих 20 слів у кожній темі. Результати наведено нижче.

In [59]:
```
for i in topic_words:
    print([str(word) for word in i])
    print()
```

```
["('love', 0.0060920287)", "('famili', 0.0057452014)", "('meet', 0.0049913
04404603)", "('marri', 0.004145138)", "('film', 0.0038797106)", "('polic',
', 0.0036526567)", "('apart', 0.0035725196)", "('two', 0.0035113371)", "('
9)", "('first', 0.0033799885)", "('kill', 0.0032587692)", "('end', 0.00322
031669671)", "('mother', 0.003145813)"]

['("n\'t", 0.0048787617)', "('home', 0.0045255101)", "('go', 0.0041958424)
5)", "('friend', 0.0037009302)", "('want', 0.0034007316)", "('arriv', 0.00
0033317441)", "('run', 0.0033157258)", "('doe', 0.0032246127)", "('life',
ney', 0.0029330065)", "('say', 0.0029074878)", "('ship', 0.0028583235)", "
1)", "('day', 0.0027676511)"]

["('kill', 0.0088955145)", "('soldier', 0.007041302)", "('order', 0.005454
1392438)", "('attack', 0.0051187645)", "('arriv', 0.0040634242)", "('die',
fight', 0.0036402682)", "('famili', 0.0035472852)", "('two', 0.0033442783)
9614)", "('offic', 0.0032748682)", "('wound', 0.0030637712)", "('water', 0
0.0028854399)", "('escap', 0.0028388696)"]

["('kill', 0.0059298733)", "('work', 0.0046990262)", "('shoot', 0.00443094
831)", "('home', 0.0036819505)", "('say', 0.0035882795)", "('day', 0.00341
133756)", "('meet', 0.0032539165)", "('town', 0.0031994844)", "('murder',
her', 0.0030694706)", "('gun', 0.0030261765)", "('friend', 0.0029322635)",
2)", "('ask', 0.0028178552)"]

["('car', 0.0076873163)", "('kill', 0.0050004474)", "('ask', 0.0049985284)
6)", "('polic', 0.0041881064)", "('call', 0.0041395766)", "('meet', 0.0040
0.0037459766)", "('drive', 0.0037118655)", "('doe', 0.0036544118)", "('day
wo', 0.0034917942)", "('friend', 0.003439873)", "('name', 0.0034290601)",
8)", "('goe', 0.0033526148)"]
```

ВИСНОВКИ ДО РОЗДІЛУ 5

У даному розділі були розглянуті основні особливості реалізації системи кластерного аналізу науково-технічних даних, а також наведено опис основних засобів розробки системи.

ВИСНОВКИ

В результаті роботи було виконано дослідження предметної області, аналіз теоретичних засад та математичних методів кластерного аналізу, проектування, реалізацію та тестування програмного додатку системи кластерного аналізу та візуалізації науково-технічних даних.

Проектування здійснювалося відповідно до парадигм процедурного та об’єктно-орієнтованого програмування. Завдяки здійсненому процесу проектування було уникнено ситуацій, пов’язаних з невизначеністю функціоналу, набором значимих сутностей програми. Також вдалося зменшити до мінімуму час, затрачений на написання програмного коду.

Розроблена система кластерного аналізу та візуалізації даних виконує поставлені перед нею задачі: аналіз даних, виконання алгоритмів кластеризаціі та візуалізація отриманих результатів.

Наступними напрямками розвитку системи можна визначити: реалізація більшої кількості алгоритмів кластеризації, розширення функціоналу для роботи з будь-якими даними, покращення продуктивності при виконанні візуалізації.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A. K. Jain, R. C. Dubes. Algorithms for Clustering Data. Prentice Hall Advanced Reference Series. Prentice Hall, March 1988.
2. A. K. Jain, R. C. Dubes. Algorithms for Clustering Data. – New Jersey: Prentice Hall, 1988. – 334p.
3. M.R. Anderberg. Cluster Analysis for Applications. Academic Press, New York, December 1973.
4. M.R. Anderberg. Cluster Analysis for Applications. - New York: Academic Press, December 1973. – 359p.
5. L. Kaufman, P.J. Rousseeuw. Finding Groups in Data: An Introdution to Cluster Analysis. Wiley Series in Probability and Statistics. John Wiley and Sons, New York, November 1990.
6. L. Kaufman, P.J. Rousseeuw. Finding Groups in Data: An Introdution to Cluster Analysis. – New York: John Wiley and Sons, November 1990. – 365p.
7. M.S. Aldenderger, R.K. Blashfield. Cluster Analysis. Sage Publication, Los Angeles, 1985.
8. M.S. Aldenderger, R.K. Blashfield. Cluster Analysis. – Los Angeles: Sage Publication, 1985. – 286p.
9. B.S. Everitt, S.Landau, M. Leese. Cluster Analysis. Arnold Publishers, London, fourth edition, May 2001.
10. B.S. Everitt, S.Landau, M. Leese. Cluster Analysis. – London: Arnold Publishers, 2001. – 343p.
11. J. Hartigan. Clustering Algorithms. Wiley, New York, 1975.
12. J. Hartigan. Clustering Algorithms. – New York, Wiley, 1975. – 402p.
13. B. Mirkin. Mathematical Classification and Clustering, volume 11 of Nonconvex Optimization and Its Application. Kluwer Academic Publishers, August 1996.

14. Mirkin. Mathematical Classification and Clustering, volume 11 of Nonconvex Optimization and Its Application. – Berlin: Kluwer Academic Publishers, 1996. – 458p.

15. Воронцов К.В. Алгоритми кластеризации и многомерного шкалирования. Курс лекций. МГУ, 2007.

16. И.А. Чубукова. Data Mining. Учебное пособие. – М: Интернет-Университет Информационных технологий; Бином. Лаборатория знаний, 2006. – 382с.

17. Машинное обучение, распознавание образов и интелектуальный анализ данных [Електронний ресурс]: Режим доступу: http://www.machinelearning.ru/ - Назва з екрану. 22.05.2020.

18. Cluster Analysis: Basic Concepts and Algorithms [Електронний ресурс]: Режим доступу: https://www-users.cs.umn.edu/~kumar/dmbook/ch8.pdf - Назва з екрану. 22.05.2020.

19. Comparison between Data Clustering Algorithms [Електронний ресурс]: Режим доступу: http://iajit.org/PDF/vol.5,no.3/15-191.pdf - Назва з екрану. 22.05.2020.

20. Програмная система кластерного анализа данных смешаного типа [Електронний ресурс]: Режим доступу: http://www.jurnal.nips.ru/sites/default/files/Paper-2013-1-11.pdf - Назва з екрану. 22.05.2020.

21. Методы и средства анализа данных [Електронний ресурс]: Режим доступу: http://bourabai.kz/tpoi/analysis6.htm - Назва з екрану. 22.05.2020.

22. Performance Analysis for Quality Measures Using K means Clustering and EM models in Segmentation of Medical Images [Електронний ресурс]:Режим доступу:http://www.ijsce.org/attachments/File/Vol-1_Issue-6/F0271111511.pdf - Назва з екрану. 22.05.2020.

Додаток 1. Специфікація

Позначення	Найменування	Примітки
Документація		
1	Пояснювальна записка	н/п
2	Додаток 2. Текст програмного модулю	н/п
3	Додаток 3. Опис програмного модулю	н/п
Компоненти		
1	Алгоритм t-SNE	Зниження розмірності, що дозволяє ефективно візуалізувати багатовимірні компоненти.
2	Метод головних компонент	
3	Словниковий метод видалення стоп-слів	Перевагою даного методу є точність. Недоліками є трудоємкість (список стоп-слів складається вручну) та недостатньо повне видалення стоп-слів (невелика універсальність словника).
4	Статистичний метод видалення стоп-слів та побудований на основі Y-інтерпретації закону Бредфорда	Метод полягає в аналізі великої кількості текстів з предметної області, до якої належить досліджуваний текст, і вибірка з цих текстів списку найбільш слів, що зустрічаються найчастіше.
5	Статистична міра TF-IDF	Використовується для оцінки важливості слова, як в контексті документа (TF), так і в контексті корпусу документів (IDF). Важлива особливість використання TF-IDF - набір даних не повинен змінюватися під час розрахунку.
7	Алгоритм K-середніх	Неієрархічна кластеризація, що базується на прототипах, що намагається знайти специфічну групу кластерів (K), які відображаються через центроїди.
8	Алгоритм ієрархічної класифікації Уорда	Ієрархічна кластеризація, що базується на техніці

		кластеризації починаючи з кожної точки як окремого кластера і далі замінюючи одинарний кластер кластером з двох найближчих точок замість однієї.
9	Алгоритм DBSCAN	Кластеризація, що базується на щільності розподілу і виконує неієрархічну кластеризацію, в якій кількість кластерів автоматично визначається алгоритмом.

Додаток 2. Текст програмного модуля головного методу

1. K-mean кластеризація

```
from sklearn.cluster import KMeans

num_clusters = 5

km = KMeans(n_clusters=num_clusters)

%time km.fit(tfidf_matrix)

clusters = km.labels_.tolist()

from sklearn.externals import joblib

#joblib.dump(km,  'doc_cluster.pkl')
km = joblib.load('doc_cluster.pkl')
clusters = km.labels_.tolist()

import pandas as pd

films = { 'назва': titles, 'ранг': ranks, 'опис': synopses, 'кластер': clusters, 'жанр': genres }

frame = pd.DataFrame(films, index = [clusters] , columns = ['ранг', 'назва', 'кластер', 'жанр'])

frame['кластер'].value_counts()

grouped = frame['ранг'].groupby(frame['кластер'])

grouped.mean()

from __future__ import print_function

print("Верхні терми для кластерів:")
print()
order_centroids = km.cluster_centers_.argsort()[:, ::-1]
for i in range(num_clusters):
    print("Кластер %d слова:" % i, end='')
    for ind in order_centroids[i, :6]:
        print(' %s' % vocab_frame.ix[terms[ind].split('
')].values.tolist()[0][0].encode('utf-8', 'ignore'), end=',')
    print()
    print()
```

```
    print("Кластер %d назви:" % i, end='')
    for title in frame.ix[i]['назва'].values.tolist():
        print(' %s,' % title, end='')
    print()
    print()

frame['Ранг'] = frame['ранг'] + 1
frame['Назва'] = frame['назва']

#експорт таблиць в HTML
print(frame[['Ранг', 'Назва']].loc[frame['кластер'] == 1].to_html(index=False))
```

2. Ієрархічна кластеризація документів

```
from scipy.cluster.hierarchy import ward, dendrogram

linkage_matrix = ward(dist) #визначаємо матрицю дистанцій за допомогою алгоритму Уорда

fig, ax = plt.subplots(figsize=(15, 20))
ax = dendrogram(linkage_matrix, orientation="right", labels=titles);

plt.tick_params(\
    axis= 'x',
    which='both',
    bottom='off',
    top='off',
    labelbottom='off')

plt.tight_layout() #будуємо дендрограму
#відкоментувати для можливості збереження діаграми у файл
plt.savefig('ward_clusters.png', dpi=200) #зберегти рисунок у файл
```

3. Прихований розподіл Діріхле

```
#видаляємо власні іменники
import string
def strip_proppers(text):
    # токенізуємо за реченнями, потім за словами
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent) if word.islower()]
    return "".join([" "+i if not i.startswith("'") and i not in string.punctuation else i for i in tokens]).strip()
```

```
#видалаямо власні іменники
from nltk.tag import pos_tag

def strip_proppers_POS(text):
    tagged = pos_tag(text.split()) #використовуємо модуль визначення частин мови NLTK
    non_propernouns = [word for word,pos in tagged if pos != 'NNP' and pos != 'NNPS']
    return non_propernouns

#Реалізація прихованого розподілу Діріхле засобами GenSim

from gensim import corpora, models, similarities

#видаляємо власні іменники
preprocess = [strip_proppers(doc) for doc in synopses]

%time tokenized_text = [tokenize_and_stem(text) for text in preprocess]

%time texts = [[word for word in text if word not in stopwords] for text in tokenized_text]

#print(len([word for word in texts[0] if word not in stopwords]))
print(len(texts[0]))

dictionary = corpora.Dictionary(texts)

dictionary.filter_extremes(no_below=1, no_above=0.8)

corpus = [dictionary.doc2bow(text) for text in texts]

len(corpus)

%time lda = models.LdaModel(corpus, num_topics=5, id2word=dictionary, update_every=5, chunksize=10000, passes=100)

print(lda[corpus[0]])

topics = lda.print_topics(5, num_words=20)

topics_matrix = lda.show_topics(formatted=False, num_words=20)

topics_matrix = np.array(topics_matrix, dtype=object)
```

```
topics_matrix.shape

topic_words = topics_matrix[:,1]

for i in topic_words:
    print([str(word) for word in i])
    print()
```

Додаток 3. Опис програмного модуля

АНОТАЦІЯ

Здійснено проектування у відповідності до парадигм процедурного та об’єктно-орієнтованого програмування. Завдяки здійсненому процесу проектування уникнено ситуацій, пов’язаних з невизначеністю функціоналу, набором значимих сутностей програми. Зменшено до мінімуму час, затрачений на написання програмного коду.

Розроблена система кластерного аналізу та візуалізації даних виконує поставлені перед нею задачі: аналіз даних, виконання алгоритмів кластеризації та візуалізація отриманих результатів.

ЗМІСТ

1 ЗАГАЛЬНІ ВІДОМОСТІ ТА ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Datamining – це технологія, яка вивчає процес отримання нових, реальних та раніше невідомих знань в базах даних. Однією із задач технології Datamining є процес кластеризації. Процес кластеризації дозволяє розглядати великі набори даних та різко скорочувати їх, робити їх компактними та наглядними. Задача кластеризації полягає в розподіленні деякої множини об’єктів на групи схожих об’єктів, що називаються кластерами. Результатом процесу кластеризації являється набір кластерів, які містять в собі схожі елементи вхідної множини елементів.

Для реалізації сервісу була обрана високорівнева мова програмування Python, реалізація інтерпретатора 2.7. Одним з її переваг є легкість для читання і ясність. Програмний код на мові Python читається легше, а значить

багаторазове його використання і обслуговування виконується набагато простіше, ніж використання програмного коду на інших мовах.

Крім того, Python підтримує сучасні механізми багаторазового використання програмного коду з арсеналу об'єктно-орієнтованого і функціонального програмування. У порівнянні зі строго типізований мовами, такими як C, C ++ і Java, Python у багато разів підвищує продуктивність праці розробника. У складі Python поставляється велика кількість зібраних і переносяться функціональних можливостей, що становлять стандартну бібліотеку.

Як середовище розробки сервісу була обрана IDE Sublime Text - досить розвинена і широко відома інтегроване середовище розробки модульних крос-платформних додатків, що є вільним програмним забезпеченням.

Sublime Text являє собою першу настільки потужно підтриману світовим IT-спільнотою спробу створення єдиної відкритої інтегрованої платформи розробки додатків, що володіє надійністю, функціональністю і рівнем якості комерційного продукту. Фактично, ця платформа призначена для всього і ні для чого конкретно, представляючи собою основу, що має блочну структуру і інтегруючу інструменти розробки ПО різних виробників для створення додатків на будь-якій мові, з використанням будь-яких технологій і для будь-якої програмної платформи.

Перевагами Sublime Text є:

1. Швидкість та низькі вимоги до ресурсів комп’ютера.
2. Робота в популярних операційних системах, таких як Windows, Linux и Mac OS.
3. Сторонні плагіни та доповнення.

Бібліотека NLTK - пакет бібліотек і програм для символьної і статистичної обробки природної мови, написаних на мові програмування Python. Містить графічні уявлення і приклади даних. Супроводжується великою кількістю документацією, включаючи книгу з поясненням основних

концепцій, що стоять за тими завданнями обробки природної мови, які можна виконувати за допомогою даного пакету.

На сьогоднішній день перед людиною стоїть проблема вибору або фільтрації необхідної інформації серед інформаційного шуму. Основну частину всієї корисної інформації на Землі складають тексти.

Для вирішення даної задачі виник напрямок інтелектуального аналізу текстів (англомовна назва Text Mining). Його відносять до Data Mining, він також включає в себе методи статистики, лінгвістики та штучного інтелекту. Вимогами до системи є точність, ефективність, продуктивність, гнучкість, стійкість, масштабованість, мультимодальність, розрідженість даних, багатомовність.

2 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

2.1 Загальні етапи

Процес аналізу текстових документів можна уявити як послідовність декількох кроків (рис. 2.1).

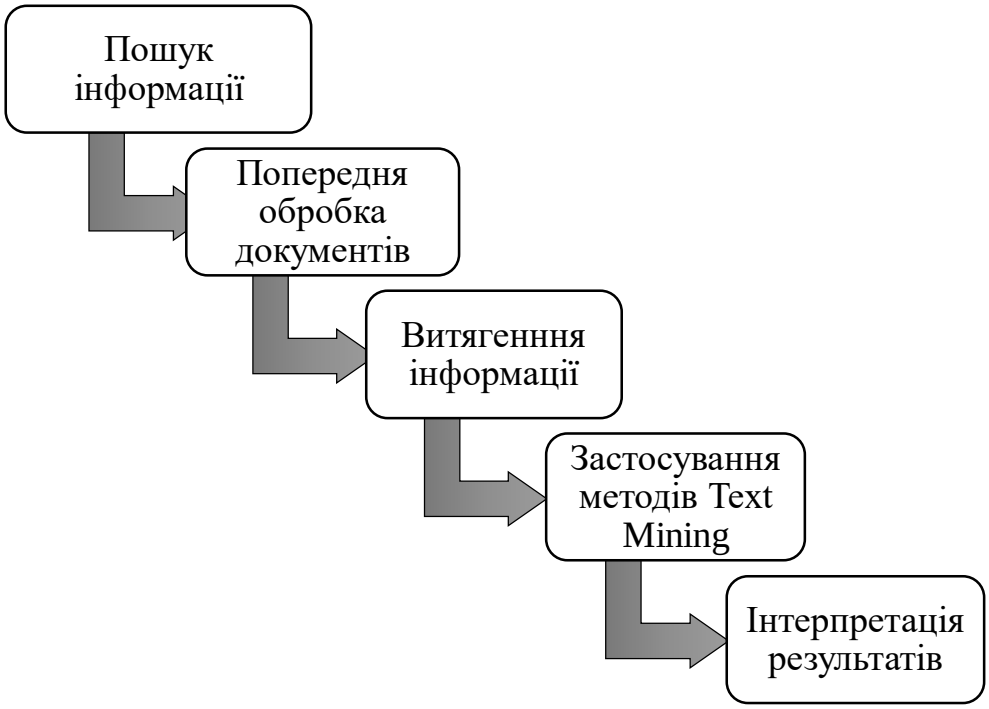


Рисунок 2.1 – Етапи Text Mining

1. *Пошук інформації*. На першому кроці необхідно ідентифікувати, які документи повинні бути проаналізовані і забезпечити їх доступність.

2. *Попередня обробка документів*. На цьому кроці виконуються найпростіші, але необхідні перетворення з документами для ставлення їх у вигляді, з яким працюють методи Text Mining. Метою таких перетворень є видалення зайвих слів і надання тексту більш строгої форми.

3. *Витягнення інформації*. Витяг інформації з обраних документів передбачає виділення в них ключових понять, над якими надалі буде виконуватися аналіз.

4. *Застосування методів Text Mining*. На даному етапі витягуються шаблони і відношення, наявні в текстах. Даний крок є основним в процесі аналізу текстів.

5. *Інтерпретація результатів*. Останній крок у процесі виявлення знань передбачає інтерпретацію отриманих результатів. як правило, інтерпретація полягає або в поданні результатів на природньою мовою, або в їх візуалізації в графічному вигляді.

2.2. Структура системи

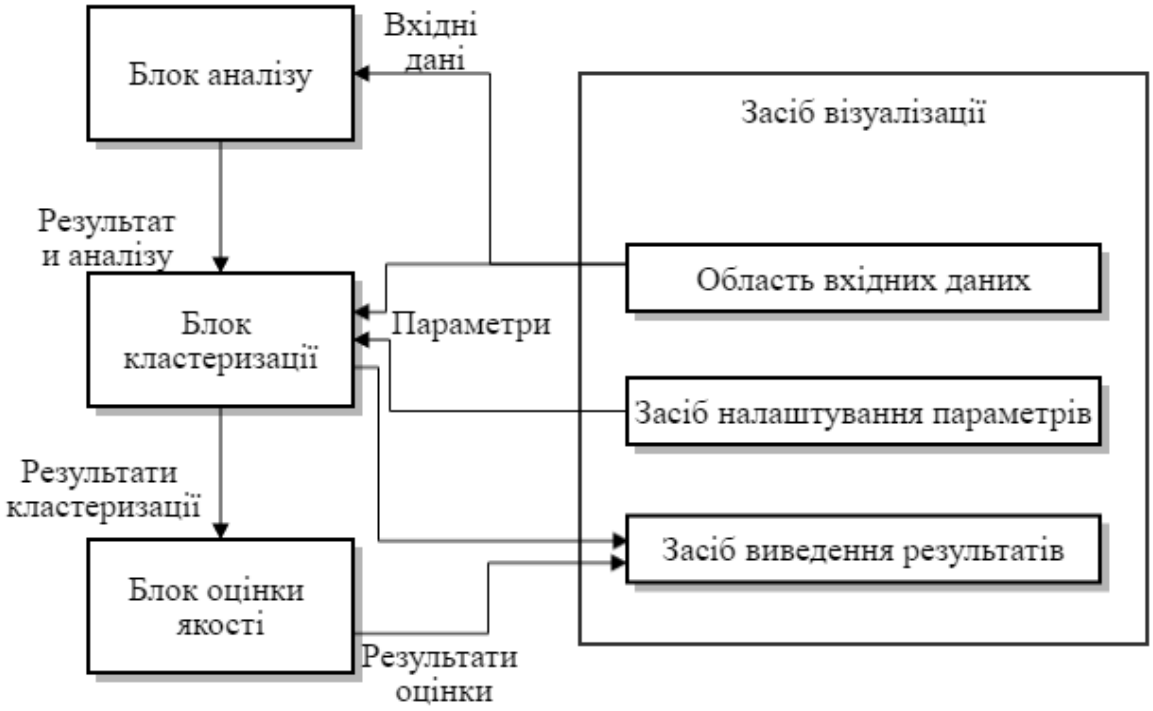


Рисунок 2.2 – Структура системи

Блок аналізу необхідний для виконання попереднього аналізу вхідних даних. А саме, перед початком кластеризації системі мають бути відомі такі відомості як кількість рядків в файлі, кількість і тип змінних (кількісний, якісний, номінальний) і т.д. Це виконується на етапі попереднього аналізу.

Блок кластеризації використовується для безпосереднього розподілу об’єктів до кластерів. В цьому блоці проводиться обчислення степені схожості між об’єктами. Метод обчислення міри схожості може різним і залежати від налаштувань системи. Після отримання матриці схожості блок кластеризації розподіляє об’єкти в кластери. Вибір алгоритма кластеризації та його параметрів здійснюється в налаштуваннях системи.

Засіб оцінки якості кластреризації призначений для оцінки ступеня достовірності кластерних рішень, на основі яких будуть видані відповідні рекомендації.

Засіб візуалізації представляє користувачу можливість взаємодії з системою.

2.3. Реалізація програмного забезпечення (приклад кластеризації наборів документів за допомогою Python)

Завантажуємо список NLTK англійських стоп-слів. Стоп-слова – це слова типу "a", "the" або "in", які не несуть для нас корисного значення:

```
In [7]: # завантажуємо nltk стоп-слова у змінну 'stopwords'
        #nltk.download()
        stopwords = nltk.corpus.stopwords.words('english')
```

Створюємо словник, який стає важливим у випадку, якщо потрібно використовувати стеми для алгоритму, але пізніше їх знову потрібно конвертувати у повні слова для цілей їх подальшого представлення.

Для виконання необхідних операцій визначимо наступні дві функції:

- *tokenize_and_stem*: розбиває опис на список відповідних слів (або лексем), а також токенізує кожну лексему;
- *tokenize_only*: токенізує лише опис.

```
In [9]: # визначаємо дві функції

def tokenize_and_stem(text):
    # токенізуємо за реченнями, далі за словами
    tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent)]
    filtered_tokens = []
    # відфільтровуємо всі токени, що не містять літер (цифрові токени, пунктуація)
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    stems = [stemmer.stem(t) for t in filtered_tokens]
    return stems


def tokenize_only(text):
    # токенізуємо за реченнями, далі за словами
    tokens = [word.lower() for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sen
    filtered_tokens = []
    # відфільтровуємо всі токени, що не містять літер (цифрові токени, пунктуація)
    for token in tokens:
        if re.search('[a-zA-Z]', token):
            filtered_tokens.append(token)
    return filtered_tokens
```

Створюємо векторизатор (tf-idf).

З метою отримання матриці Tf-idf потрібно спочатку порахувати входження слів за кожним документом. Це дозволяє отримати матрицю документ-терм (dtm) або матрицю частот термінів (див. рис.2.3).

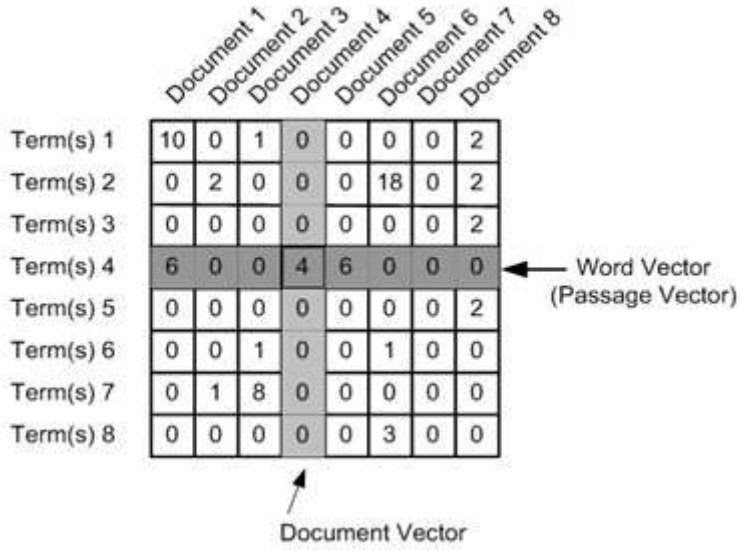


Рисунок 2.3 – Матриця частот термінів

Потім застосовуємо зважування частоти терму до зворотньої частоти документа: слова, які часто зустрічаються в документі, але не часто в корпусі, отримують більш високу вагу, оскільки вважається, що ці слова мають більше значення для документа:

- max_df: це максимальна частота в документах, яка може використовуватись у матриці tfi-idf . Якщо частота появи терму перевищує 80% документів, то він ймовірно є незначним;
- min_idf: число, яке містить кількість документів, в яку повинен входити відповідний терм;

```
In [12]: from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=200000,
                                   min_df=0.2, stop_words='english',
                                   use_idf=True, tokenizer=tokenize_and_stem, r

%time tfidf_matrix = tfidf_vectorizer.fit_transform(synopses)

print(tfidf_matrix.shape)

Wall time: 10.4 s
(100, 563)
```

Дивимося *терми* – це перелік функцій, які використовуються в матриці tf-idf:

```
In [13]: terms = tfidf_vectorizer.get_feature_names()
```

Використовуємо алгоритм K-mean. Для даного алгоритму заздалегідь визначається кількість кластерів (в даному прикладі обрано 5). Кожне спостереження присвоюється якомусь кластеру таким чином, щоб мінімізувати внутрішню кластерну суму квадратів. Далі розраховується середнє значення кластерних спостережень і використовується як новий центроїд кластера. Потім спостереження переназначаються кластерам та центроїдам, які перераховуються в ітераційному процесі, поки алгоритм не досягне конвергенції:

```
In [15]: from sklearn.cluster import KMeans

num_clusters = 5

km = KMeans(n_clusters=num_clusters)

%time km.fit(tfidf_matrix)

clusters = km.labels_.tolist()

Wall time: 504 ms
```

Використовуємо бібліотеку joblib.dump, яка дозволяє підбирати модель, як тільки вона досягає конвергенції, а також перезавантажувати її і перепризначити мітки новим кластерам:

```
In [16]: from sklearn.externals import joblib

         #joblib.dump(km,  'doc_cluster.pkl')
         km = joblib.load('doc_cluster.pkl')
         clusters = km.labels_.tolist()
```

Проіндексуємо та відсортуємо за кожним кластером, щоб визначити, які n є часто використовуваними ($n = 6$) словами, найближчими до центру кластера:

```
In [20]: from __future__ import print_function

         print("Верхні терми для кластерів:")
         print()
         order_centroids = km.cluster_centers_.argsort()[:, ::-1]
         for i in range(num_clusters):
             print("Кластер %d слова:" % i, end='')
             for ind in order_centroids[i, :6]:
                 print(' %s' % vocab_frame.ix[terms[ind].split(' ')].values.tolist()[0][0].encode('utf-8', 'ignor
             print()
             print()
             print("Кластер %d назви:" % i, end='')
             for title in frame.ix[i]['назва'].values.tolist():
                 print(' %s,' % title, end='')
             print()
             print()
```

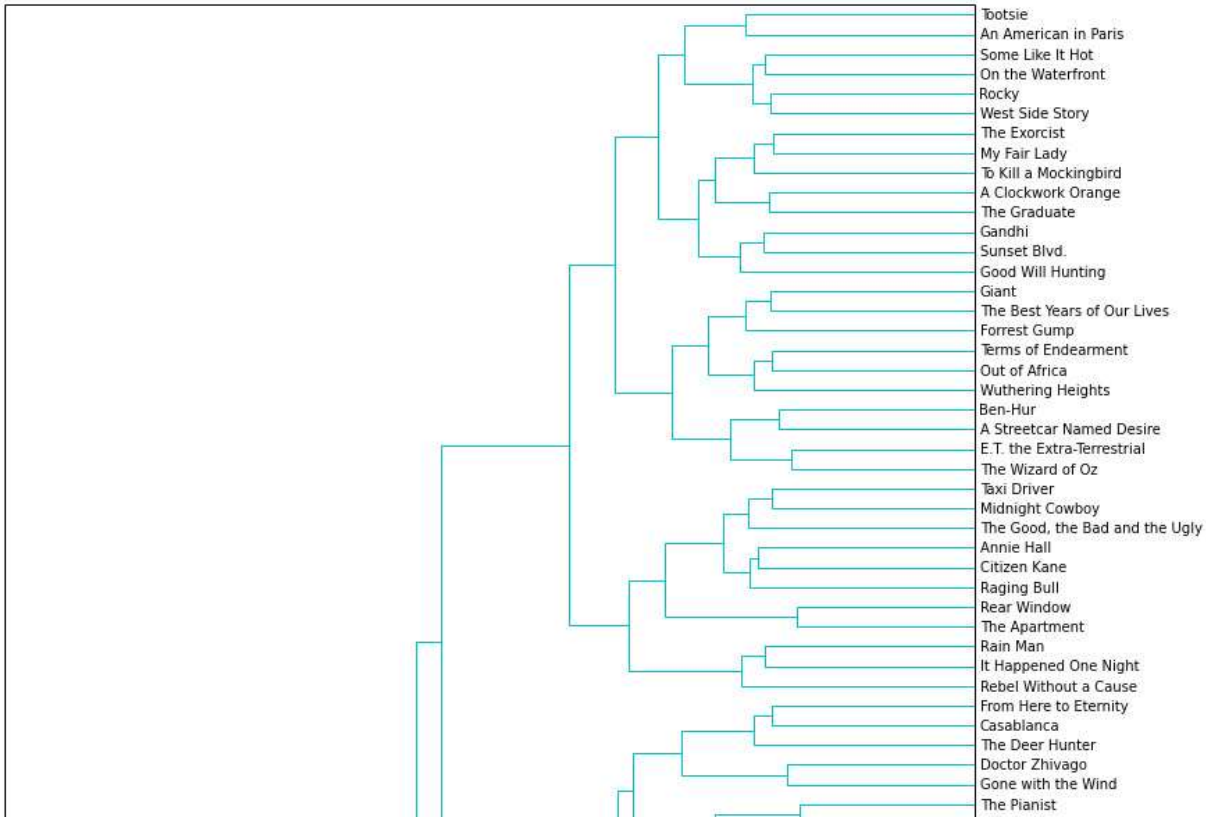
```
Верхні терми для кластерів:

Кластер 0 слова: b'family', b'home', b'mother', b'war', b'house', b'dies',
```

Перетворюємо матрицю dist у двовимірний масив, використовуючи багатовимірне масштабування:

```
In [23]: import os  # for os.path.basename

         import matplotlib.pyplot as plt
         import matplotlib as mpl

         from sklearn.manifold import MDS

         MDS()

         mds = MDS(n_components=2, dissimilarity="precomputed", random_state=1)

         pos = mds.fit_transform(dist)  # shape (n_components, n_samples)

         xs, ys = pos[:, 0], pos[:, 1]
```

Використовуємо ієрархічний алгоритм кластеризації Уорда – агломераційного методу кластеризації, який визначає, що на кожному етапі пари кластерів з мінімальними відстанями між кластерами об'єднуються. В даному прикладі використовується матриця косинусних відстаней:

```
In [31]: from scipy.cluster.hierarchy import ward, dendrogram

         linkage_matrix = ward(dist) #визначаємо матрицю дистанцій за допомогою алгоритму Уорд

         fig, ax = plt.subplots(figsize=(15, 20))
         ax = dendrogram(linkage_matrix, orientation="right", labels=titles);

         plt.tick_params(\
             axis= 'x',
             which='both',
             bottom='off',
             top='off',
             labelbottom='off')

         plt.tight_layout() #будуємо дендрограму
         #відкоментувати для можливості збереження діаграми у файл
```



Використовуємо прихований розподіл Діріхле (LDA), який дозволяє дізнатися більше про приховану структуру. LDA - це імовірнісна тематична модель, яка передбачає, що документи є сумішшю тем і що кожне слово в документі можна віднести до тем документа.

Для реалізації LDA використовуємо пакет GenSim. Нижче наведено функцію, яка видаляє в описах будь-які власні іменники:

```
In [32]: #видаляємо власні іменники
         import string
         def strip_proppers(text):
             # токенізуємо за реченнями, потім за словами
             tokens = [word for sent in nltk.sent_tokenize(text) for word in nltk.word_tokenize(sent) if word.is
             return "".join([" "+i if not i.startswith("'") and i not in string.punctuation else i for i in toke
```

Оскільки вищевказана функція ґрунтується на використанні великих літер, вона схильна видаляти слова на початку речень. Інший варіант функції без цієї вади наведений нижче і використовує модуль визначення частин мови NLTK:

```
In [33]: #видалаямо власні іменники
         from nltk.tag import pos_tag

         def strip_proppers_POS(text):
             tagged = pos_tag(text.split()) #використовуємо модуль визначення частин мови NLT
             non_propernouns = [word for word,pos in tagged if pos != 'NNP' and pos != 'NNPS'
             return non_propernouns
```

Виконуємо обробку тексту (видалення власних іменників, токенізація, видалення стоп-слів):

```
In [35]: #Реалізація прихованого розподілу Дірихле засобами GenSim

         from gensim import corpora, models, similarities

         #видаляємо власні іменники
         preprocess = [strip_proppers(doc) for doc in synopses]

         %time tokenized_text = [tokenize_and_stem(text) for text in preprocess]

         %time texts = [[word for word in text if word not in stopwords] for text in tokenized_tex
         Wall time: 5.36 s
         Wall time: 544 ms
```

```
In [37]: dictionary = corpora.Dictionary(texts)

In [38]: dictionary.filter_extremes(no_below=1, no_above=0.8)

In [39]: corpus = [dictionary.doc2bow(text) for text in texts]
```

Для створення моделі обрано 100 проходів для забезпечення конвергенції. Як можна побачити, для роботи машини знадобилося 1 хвилина 18 секунд.

```
In [41]: %time lda = models.LdaModel(corpus, num_topics=5, id2word=dict
         Wall time: 1min 18s
```

Кожна тема має набір слів, що визначає її разом з певною вірогідністю.

Перетворимо список найкращих 20 слів у кожній темі. Результати наведено нижче.

```
In [59]: for i in topic_words:
             print([str(word) for word in i])
             print()
```

```
["('love', 0.0060920287)", "('famili', 0.0057452014)", "('meet', 0.0049913
04404603)", "('marri', 0.004145138)", "('film', 0.0038797106)", "('polic',
', 0.0036526567)", "('apart', 0.0035725196)", "('two', 0.0035113371)", "('
9)", "('first', 0.0033799885)", "('kill', 0.0032587692)", "('end', 0.00322
031669671)", "('mother', 0.003145813)"]

['("n\'t", 0.0048787617)', "('home', 0.0045255101)", "('go', 0.0041958424)
5)", "('friend', 0.0037009302)", "('want', 0.0034007316)", "('arriv', 0.00
0033317441)", "('run', 0.0033157258)", "('doe', 0.0032246127)", "('life',
ney', 0.0029330065)", "('say', 0.0029074878)", "('ship', 0.0028583235)", "
1)", "('day', 0.0027676511)"]

["('kill', 0.0088955145)", "('soldier', 0.007041302)", "('order', 0.005454
1392438)", "('attack', 0.0051187645)", "('arriv', 0.0040634242)", "('die',
fight', 0.0036402682)", "('famili', 0.0035472852)", "('two', 0.0033442783)
9614)", "('offic', 0.0032748682)", "('wound', 0.0030637712)", "('water', 0
0.0028854399)", "('escap', 0.0028388696)"]

["('kill', 0.0059298733)", "('work', 0.0046990262)", "('shoot', 0.00443094
831)", "('home', 0.0036819505)", "('say', 0.0035882795)", "('day', 0.00341
133756)", "('meet', 0.0032539165)", "('town', 0.0031994844)", "('murder',
her', 0.0030694706)", "('gun', 0.0030261765)", "('friend', 0.0029322635)",
2)", "('ask', 0.0028178552)"]

["('car', 0.0076873163)", "('kill', 0.0050004474)", "('ask', 0.0049985284)
6)", "('polic', 0.0041881064)", "('call', 0.0041395766)", "('meet', 0.0040
0.0037459766)", "('drive', 0.0037118655)", "('doe', 0.0036544118)", "('day
wo', 0.0034917942)", "('friend', 0.003439873)", "('name', 0.0034290601)",
8)", "('goe', 0.0033526148)"]
```

3 ВИКЛИК І ЗАВАНТАЖЕННЯ

Для інсталяції нашої програми завантажуємо IPython та Jupyter Notebook.

IPython – являє собою потужний інструмент для роботи з мовою Python. Базові компоненти IPython – це інтерактивна оболонка з широким набором можливостей та ядро для Jupyter. IPython дозволяє підключати безліч клієнтів до одного обчислювального ядра. Завдяки своїй архітектурі, може працювати в паралельному кластері.

Jupyter Notebook – є графічною веб-оболонкою для IPython, яка розширює ідею консольного підходу до інтерактивних обчислень.

Jupyter Notebook дозволяє розробляти, документувати та запускати програми на мові Python. Він складається з двох компонентів: веб-додаток, що запускається в браузері, і ноутбук-файли, в яких можливо працювати з вихідним кодом програми, запускати його, вводити і виводити дані.

Основні відмінні риси даної платформи – це комплексна інтроспекція об’єктів, збереження історії введення протягом усіх сеансів, кешування вихідних результатів, що розширюється до системи «магічних» команд, логування сесії, додатковий командний синтаксис, підсвічування коду, доступ до системної оболонки, стикування з pdb відладчиком і Python профайлером.

Jupyter Notebook входить до складу Anaconda. Для встановлення пакету, попередньо завантажуємо програму за посиланням: www.anaconda.com.

4 ВХІДНІ ТА ВИХІДНІ ДАНІ

4.1 Вхідні дані

1) Імпортуємо всі необхідні для роботи бібліотек:

```
In [4]: import numpy as np
        import pandas as pd
        import nltk
        import re
        import os
        import codecs
        from sklearn import feature_extraction
        import mpld3
```

2) Використовуємо два основних списків:

- *'titles'*: назви фільмів у заданому порядку;
- *'synopses'*: описи фільмів, що відповідають порядку назв.

Першочергове значення має список *'synopses'* ; *'titles'* в основному використовуються для маркування.

In [3]:
```
#імпорт 3-х списків: назви, посилання і описи
titles = open('title_list.txt').read().split('\n')
#зчитуємо перші 100
titles = titles[:100]

links = open('link_list_imdb.txt').read().split('\n')
links = links[:100]

synopses_wiki = open('synopses_list_wiki.txt').read().split('\n BREAKS HERE')
synopses_wiki = synopses_wiki[:100]

synopses_clean_wiki = []
for text in synopses_wiki:
    text = BeautifulSoup(text, 'html.parser').getText()
    #видаляємо html форматування та перетворимо в unicode
    synopses_clean_wiki.append(text)

synopses_wiki = synopses_clean_wiki


genres = open('genres_list.txt').read().split('\n')
genres = genres[:100]

print(str(len(titles)) + ' назв')
print(str(len(links)) + ' посилань')
print(str(len(synopses_wiki)) + ' описів')
print(str(len(genres)) + ' жанрів')
```

```
100 назв
100 посилань
100 описів
100 жанрів
```

4.2 Вихідні дані

Створюємо словник назв, рангів, описів, кластерного призначення та жанрів. Далі, цей словник перетворюємо у Pandas DataFrame для легкого доступу:

In [17]:
```
import pandas as pd

films = { 'назва': titles, 'ранг': ranks, 'опис': synopses, 'кластер': clusters, 'жанр': genres }

frame = pd.DataFrame(films, index = [clusters] , columns = ['ранг', 'назва', 'кластер', 'жанр'])
```

In [18]:
```
frame['кластер'].value_counts()
```

Out[18]:
```
4    26
0    25
2    21
1    16
3    12
Name: кластер, dtype: int64
```

In [19]:
```
grouped = frame['ранг'].groupby(frame['кластер'])

grouped.mean()
```

Out[19]:
```
кластер
0    47.200000
1    58.875000
2    49.380952
3    54.500000
4    43.730769
Name: ранг, dtype: float64
```

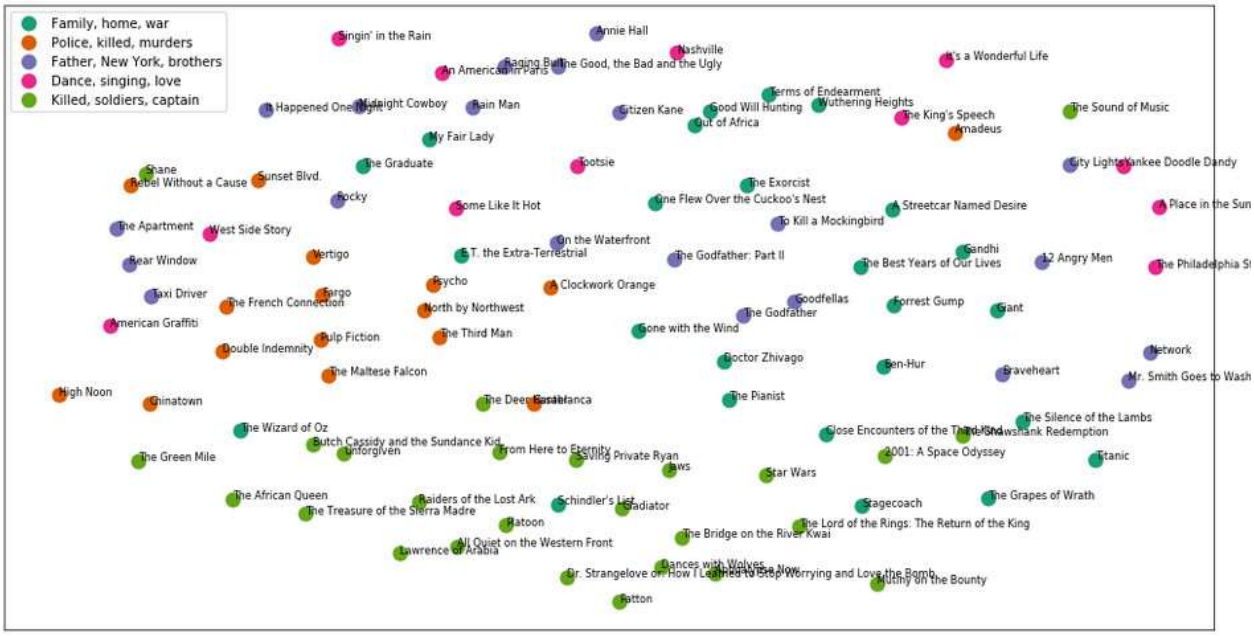
Зауважимо, що кластери 4 і 0 мають найнижчий ранг, що вказує на те, що вони, в середньому, містять фільми, які потрапили до "кращих" у топ-100 списку.

Візуалізуємо кластеризацію документів за допомогою matplotlib та mpld3 (обгортка matplotlib для D3.js).

Спочатку визначаємо деякі словники для переходу від номера кластера до кольору та до імені кластера. Назви кластерів обрано на базі слів, найближчих до кожного центру кластера.

In [25]:
```
#задаємо кольори для кластерів через словник
cluster_colors = {0: '#1b9e77', 1: '#d95f02', 2: '#7570b3', 3: '#e7298a', 4: '#66a6

#задаємо назви кластерів через словник
cluster_names = {0: 'Family, home, war',
                 1: 'Police, killed, murders',
                 2: 'Father, New York, brothers',
                 3: 'Dance, singing, love',
                 4: 'Killed, soldiers, captain'}
```

In [27]:
```
#створюємо фрейм даних
df = pd.DataFrame(dict(x=xs, y=ys, label=clusters, title=titles))

#групуємо за кластерами
groups = df.groupby('label')


# створюємо область рисування
fig, ax = plt.subplots(figsize=(17, 9)) # задаємо її розмір
ax.margins(0.05)

#ітеруємо групами для створення рисунку
for name, group in groups:
    ax.plot(group.x, group.y, marker='o', linestyle='', ms=12, label=cluster_names[name],
    ax.set_aspect('auto')
    ax.tick_params(\
        axis= 'x',
        which='both',
        bottom='off',
        top='off',
        labelbottom='off')
    ax.tick_params(\
        axis= 'y',
        which='both',
        left='off',
        top='off',
        labelleft='off')

ax.legend(numpoints=1)  #відтворюємо легенду

#додаємо мітки в x,y позиції з міткою назвою фільму
for i in range(len(df)):
    ax.text(df.ix[i]['x'], df.ix[i]['y'], df.ix[i]['title'], size=8)



plt.show() #відображаємо рисунок

#відкоментувати для можливості збереження рисунку у файл
#plt.savefig('clusters_small_noaxes.png', dpi=200)
```



In [52]:
```
plt.close()
```

Для прибирання накладок використаємо обгортку для matplotlib. Вона має гарний функціонал для збільшення та панорамування. Нижче наведено Javascript сніппет, який визначає для користувача зону відображення з можливостями зуму/панорамування:

In [29]:
```
#визначаємо клієнтське місцезнаходження тулбару
class TopToolbar(mpld3.plugins.PluginBase):
    """Plugin for moving toolbar to top of figure"""

    JAVASCRIPT = """
    mpld3.register_plugin("toptoolbar", TopToolbar);
    TopToolbar.prototype = Object.create(mpld3.Plugin.prototype);
    TopToolbar.prototype.constructor = TopToolbar;
    function TopToolbar(fig, props){
        mpld3.Plugin.call(this, fig, props);
    };

    TopToolbar.prototype.draw = function(){
      // the toolbar svg doesn't exist
      // yet, so first draw it
      this.fig.toolbar.draw();

      // then change the y position to be
      // at the top of the figure
      this.fig.toolbar.toolbar.attr("x", 150);
      this.fig.toolbar.toolbar.attr("y", 400);

      // then remove the draw function,
      // so that it is not called again
      this.fig.toolbar.draw = function() {}
    }
    """
    def __init__(self):
        self.dict_ = {"type": "toptoolbar"}
```

Для стилювання шрифту, осей та лівого поля на рисунку використовуємо mpld3, який дозволяє визначити деякий користувальницький CSS:

In [30]:

```
#створюємо фрейм даних
df = pd.DataFrame(dict(x=xs, y=ys, label=clusters, title=titles))

#групуємо за кластерами
groups = df.groupby('label')

#визначаємо користувацький css для форматування шрифту та видалення назв осей
css = """
text.mpld3-text, div.mpld3-tooltip {
  font-family:Arial, Helvetica, sans-serif;
}

g.mpld3-xaxis, g.mpld3-yaxis {
display: none; }
"""

# Рисунок
fig, ax = plt.subplots(figsize=(14,6)) #задаємо розмір рисунку
ax.margins(0.03)

for name, group in groups:
    points = ax.plot(group.x, group.y, marker='o', linestyle='', ms=18, label=clu
    ax.set_aspect('auto')
    labels = [i for i in group.title]

    #задаємо підказки використовуючи точки, назви та вже визначений 'css'
    tooltip = mpld3.plugins.PointHTMLTooltip(points[0], labels,
                                       voffset=10, hoffset=10, css=css)
    #звязуємо підказки з рисунком
    mpld3.plugins.connect(fig, tooltip, TopToolbar())

    ax.axes.get_xaxis().set_ticks([])
    ax.axes.get_yaxis().set_ticks([])

    #виключаємо назви осей
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)


ax.legend(numpoints=1) #відображаємо легенду

mpld3.display() #відображаємо створений рисунок

#відкоментувати для експорту в html
#html = mpld3.fig_to_html(fig)
#print(html)
```

Out[30]: